ZEAL EDUCATION SOCIETY's

ZEAL COLLEGE OF ENGINEEIRNG AND RESEARCH, NARHE, PUNE

DEPARTMENT OF COMPUTER ENGINEERING

SEMESTER-I

[A.Y.: 2022 - 2023]

Laboratory Practice III
(410246)
Design and Analysis of Algorithms

LABORATORY MANUAL

INSTITUTE VISION	To impart value added technological education through pursuit of academic excellence, research and entrepreneurial attitude.
INSTITUTE MISSION	**M1:** To achieve academic excellence through innovative teaching and learning process. **M2:** To imbibe the research culture for addressing industry and societal needs**.** **M3:** To provide conducive environment for building the entrepreneurial skills. **M4:** To produce competent and socially responsible professionals with core human values.

DEPARTMENT VISION	To emerge as a department of repute in Computer Engineering which produces competent professionals and entrepreneurs to lead technical and betterment of mankind.
DEPARTMENT MISSION	**M1:** To strengthen the theoretical and practical aspects of the learning process by teaching applications and hands on practices using modern tools and FOSS technologies. **M2:** To endavor innovative interdisciplinary research and entrepreneurship skills to serve the needs of Industry and Society. **M3:** To enhance industry academia dialog enabling students to inculcate professional skills. **M4:** To incorporate social and ethical awareness among the students to make them conscientious professionals.

PEO1: To Impart fundamentals in science, mathematics and engineering to cater the needs of society and Industries.
PEO2: Encourage graduates to involve in research, higher studies, and/or to become entrepreneurs.
PEO3: To Work effectively as individuals and as team members in a multidisciplinary environment with high ethical values for the benefit of society.

Savitribai Phule Pune University

Third Year of Computer Engineering (2019 Course)
410246: Laboratory Practice III

Teaching Scheme:	Credit	Examination Scheme:
PR: 04 Hours/Week	02	TW: 50 Marks
		PR: 50 Marks

Course Objectives:

Learn effect of data preprocessing on the performance of machine learning algorithms
Develop in depth understanding for implementation of the regression models
Implement and evaluate supervised and unsupervised machine learning algorithms

Course Outcomes:

On completion of the course, student will be able to-

- **Machine Learning:**

  CO1: Apply preprocessing techniques on datasets

  CO2: Implement and evaluate linear regression and random forest regression models.

  CO3: Apply and evaluate classification and clustering techniques

- **Design and Analysis of Algorithms:**

  CO4: Analyze performance of an algorithm.

  CO5: Learn how to implement algorithms that follow algorithm design strategies namely divide and conquer, greedy, dynamic programming, backtracking, branch and bound.

- **Block chain Technology:**

  CO6: Interpret the basic concepts in Block chain technology and its applications.

Sr. No.	TITLE
	Group A
01	Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.
02	Write a program to implement Huffman Encoding using a greedy strategy.
03	Write a program to solve a fractional Knapsack problem using a greedy method.
04	Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.
05	Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen_s matrix.

Aim: Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

Objective: Student will be able to learn
1. Concept of Fibonacci Numbers.

2. Analysis of time complexity in Non-Recursive and Recursive way.

Theory:
The Fibonacci numbers are the numbers in the following integer sequence.
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,……..
In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation
Fn = Fn-1 + Fn-2
With seed values
F0 = 0 and F1 = 1.
Non-Recursive Algorithm:

```
1 Algorithm fibonacci(n)
2 //Calculate nth fibonacci number
3 {
4 if(n<=1)
5 write (n);
6 else
7 {
8 fnm2:=0; fnm1:=1;
9 for i:=2 to n do
10 { fn:=fnm1+fnm2;
11 fnm2:=fnm1;
12 fnm1:=fn;
13 }
14 write(fn)
15 }
16 }
```

Time Complexity:

Two cases (1) n = 0 or 1 and (2) n>1.

1. When n = 0 or 1, lines 4 and 5 get executed once each. Since each line has an s/e of 1, the total step count for this case is 2.
2. When n > 1, lines 4, 8, and 14 are each executed once. Line 9 gets executed n times, and lines 11 and 12 get executed n-1 times each. Line 8 has an s/e of 2, line 12 has an s/e of 2, and line 13 has an s/e of 0. The remaining lines that get executed have s/e‘s of 1. The total steps for the case n > 1 is therefore 4n + 1.

Recursive Algorithm:

```
Algorithm Fibonacci(n)
{
if (n <= 1)
return n;
else
return Fibonacci(n - 1) + Fibonacci(n - 2); }
```

Time complexity:

T(n) = T(n-1) + T(n-2) + c
= 2T(n-1) + c //from the approximation T(n-1) ~ T(n-2)
= 2*(2T(n-2) + c) + c
= 4T(n-2) + 3c
= 8T(n-3) + 7c
= 2 k * T(n - k) + (2 k - 1)*c
Let's find the value of k for which: n - k = 0
k = n
T(n) = 2 n * T(0) + (2 n - 1)*c
= 2 n * (1 + c) - c
T(n) = 2 n

Conclusion: We have studied Recursive and Non-Recursive way to Calculate Fibonacci Numbers.

Aim: Write a program to implement Huffman Encoding using a greedy strategy.

Objective: Student will be able to learn
1. Concept of Huffman Encoding

2. Analysis of time complexity

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.
The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.
Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be ―cccd‖ or ―ccb‖ or ―acd‖ or ―ab‖.
There are mainly two major parts in Huffman Coding
1. Build a Huffman Tree from input characters.
2. Traverse the Huffman Tree and assign codes to characters.
Steps to build Huffman Tree
Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.
i) Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root

ii) Extract two nodes with the minimum frequency from the min heap.

iii) 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.

iv) Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Let us understand the algorithm with an example:
character Frequency

a 5
b 9
c 12
d 13
e 16
f 45
character Frequency
a 5
b 9
c 12
d 13
e 16
f 45

Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.

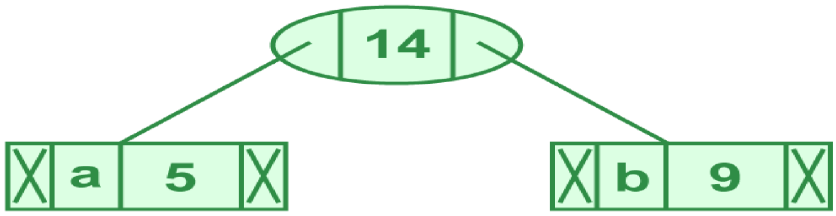


Illustration of step 2

Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements
character Frequency
c 12
d 13
Internal Node 14
e 16
f 45

Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25

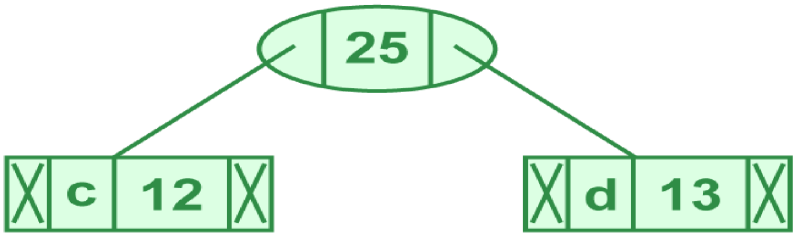


Illustration of step 3

Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes
character Frequency
Internal Node 14
e 16
Internal Node 25
f 45

Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30

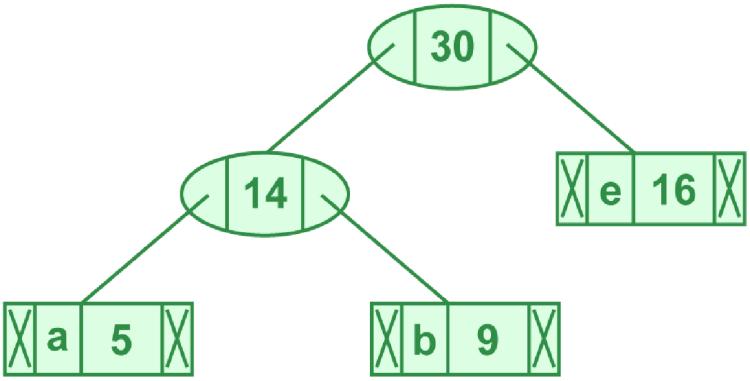


Illustration of step 4

Now min heap contains 3 nodes.
character Frequency
Internal Node 25
Internal Node 30
f 45

Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55

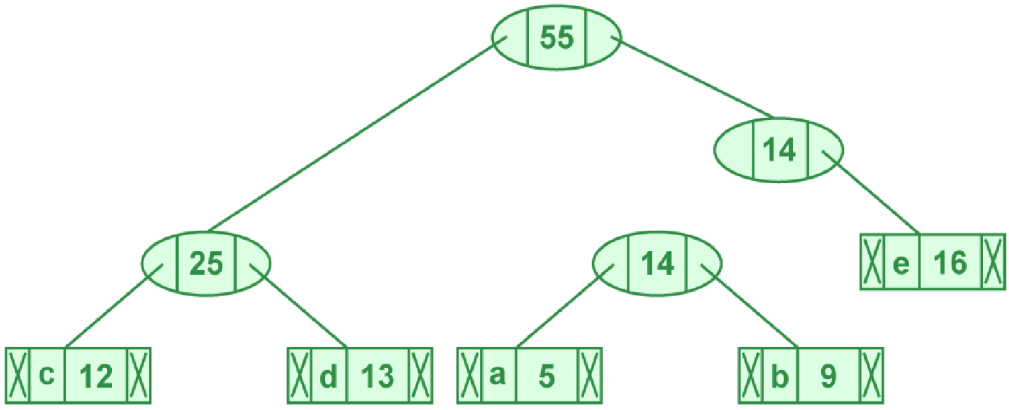


Illustration of step 5

Now min heap contains 2 nodes.
character Frequency
f 45
Internal Node 55

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100

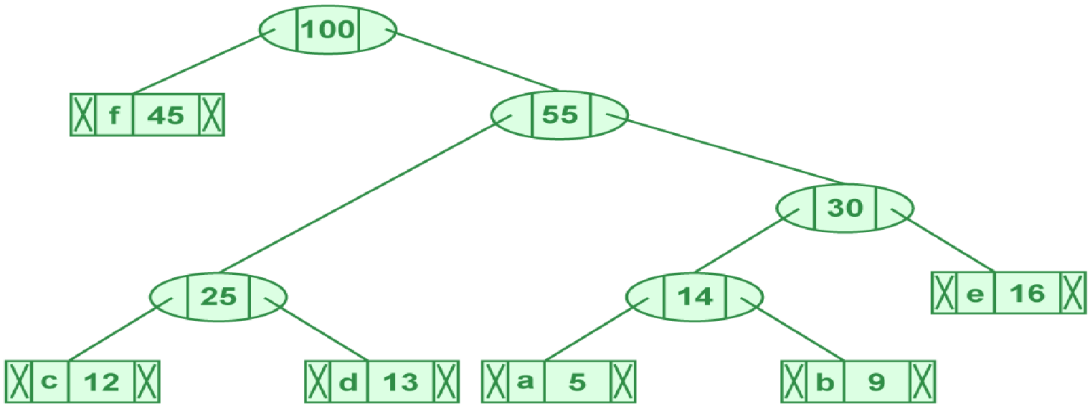
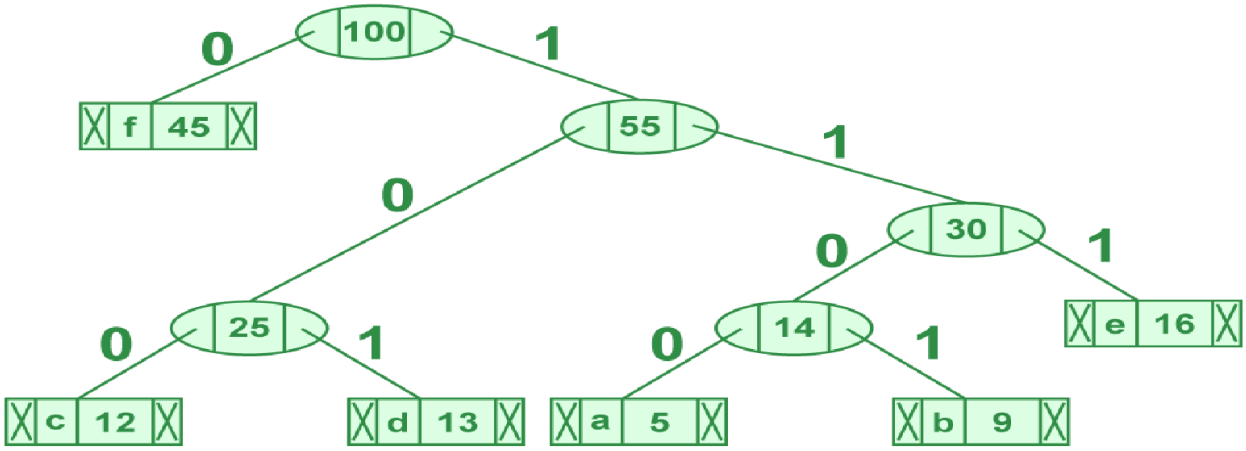


Illustration of step 6

Now min heap contains only one node.
character Frequency
Internal Node 100
Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



Steps to print code from Huffman Tree

codes are as follows:
character code-word
f 0
c 100
d 101
a 1100
b 1101
e 111
Time complexity: O(nlogn) where n is the number of unique characters.

Conclusion: We have studied Huffman encoding using Greedy Method.

Aim: Write a program to solve a fractional Knapsack problem using a greedy method.

Objective: Student will be able to learn
1. Concept of Knapsack Problem

2. Concept of implementation of Fractional Knapsack using Greedy Strategy.

Theory:
Given the 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 Fractional Knapsack, we can break items for maximizing the total value of the knapsack
Note: In the 0-1 Knapsack problem, we are not allowed to break items. We either take the whole item or don't takes it.
An efficient solution is to use the Greedy approach. The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on the basis of this ratio. Then take the item with the highest ratio and add them until we can 't adds the next item as a whole and at the end add the next item as much as we can which will always be the optimal solution to this problem.

Algorithm GreedyKnapsack(M,N)
```
//p[1:n] and w[1:n] contain the profits and weights respectively
//of the n objects ordered such that p[i]/w[i]>=p[i+1]/w[i+1]
//m is the knapsack size and x[1:n] is the solution vector.
{ for i:=1 to n do x[i]:=0.0; // initialize x
U:=m;
for i:=1 to n do
{ if (w[i]>U) then break;
x[i]:=1.0;
U=U-w[i];
} if(i<=n) then x[i]:=U/w[i];
}
```
Analysis:
The main time taking step is the sorting of all items in decreasing order of their profit/ weight ratio.
If the items are already arranged in the required order, then while loop takes O(n) time.
The average time complexity of Quick Sort is O(nlogn).
Therefore, total time taken including the sort is O(nlogn).

Conclusion: We have studied to implement fractional Knapsack using Greedy Strategy.

Aim: Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

Objective: Students will learn

1. The basic concept of 0/1 Knapsack Problem

2. The basic concept of 0/1 Knapsack using Dynamic Programming and Branch and Bound Approach.

Theory:

Knapsack Problem:

Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Dynamic Programming Approach

Let *i* be the highest-numbered item in an optimal solution S for W dollars. Then $S' = S - \{i\}$ is an optimal solution for $W - wi$ dollars and the value to the solution S is Vi plus the value of the sub-problem.

We can express this fact in the following formula: define c[i, w] to be the solution for items 1,2, … , i and the maximum weight w.

The algorithm takes the following inputs

- The maximum weight W
- The number of items n
- The two sequences v = <v1, v2, …, vn> and w = <w1, w2, …, wn>

Algorithm

Dynamic-0-1-knapsack (v, w, n, W)
for w = 0 to W do
c[0, w] = 0
for i = 1 to n do
c[i, 0] = 0
for w = 1 to W do
if wi ≤ w then
if vi + c[i-1, w-wi] then
c[i, w] = vi + c[i-1, w-wi]
else c[i, w] = c[i-1, w]
else

c[i, w] = c[i-1 set of items to take can be deduced from the table, starting at c[n, w] and tracing backwards where the optimal values came from.

If *c[i, w] = c[i-1, w]*, then item *i* is not part of the solution, and we continue tracing with c[i-1, w]. Otherwise, item *i* is part of the solution, and we continue tracing with c[i-1, w-W].

Analysis

This algorithm takes θ(*n*, *w*) times as table *c* has (*n* + 1).(*w* + 1) entries, where each entry requires θ(1) time to compute.

Branch and Bound Approach:

Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. Branch and Bound solve these problems relatively quickly.

Let us consider below 0/1 Knapsack problem to understand Branch and Bound. Consider an example where *n* = 4, and the values are given by {10, 12, 12, 18}and the weights given by {2, 4, 6, 9}. The maximum weight is given by *W = 15*. Here, the solution to the problem will be including the first, third and the fourth objects. In solving this problem, we shall use the Least Cost- Branch and Bound method, since this shall help us eliminate exploring certain branches of the tree. We shall also be using the fixed-size solution here. Another thing to be noted here is that this problem is a maximization problem, whereas the Branch and Bound method is for minimization problems. Hence, the values will be multiplied by -1 so that this problem gets converted into a minimization problem.

Now, consider the 0/1 knapsack problem with *n* objects and total weight W. We define the upper bound(U) to be the summation of vixi (where vi denotes the value of that objects, and xi is a binary value, which indicates whether the object is to be included or not), such that the total weights of the included objects is less than *W*. The initial value of U is calculated at the initial position, where objects are added in order until the initial position is filled.

We define the cost function to be the summation of vifi, such that the total value is the maximum that can be obtained which is less than or equal to *W*. Here fi indicates the fraction of the object that is to be included. Although we use fractions here, it is not included in the final solution.

Here, the procedure to solve the problem is as follows are:

● Calculate the cost function and the Upper bound for the two children of each node. Here, the (i + 1)th level indicates whether the ith object is to be included or not.

● If the cost function for a given node is greater than the upper bound, then the node need not be explored further. Hence, we can kill this node. Otherwise, calculate the upper bound for this node. If this value is less than *U*, then replace the value of *U* with this value. Then, kill all unexplored nodes which have cost function greater than this value.

● The next node to be checked after reaching all nodes in a particular level will be the one with the least cost function value among the unexplored nodes.

● While including an object, one needs to check whether the adding the object crossed the threshold. If it does, one has reached the terminal point in that branch, and all the succeeding objects will not be included.

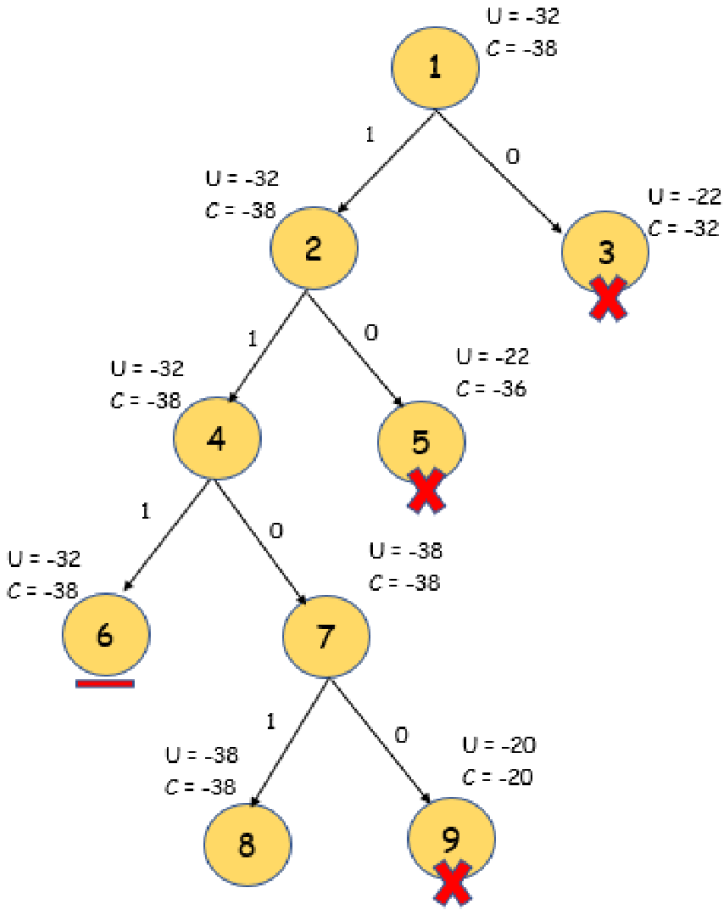
In this manner, we shall find a value of U at the end which eliminates all other possibilities. The path to this node will determine the solution to this problem.

Time and Space Complexity:

Even though this method is more efficient than the other solutions to this problem, its worst-case time complexity is still given by O(2n), in cases where the entire tree has to be explored. However, in its best case, only one path through the tree will have to explored, and hence its best-case time complexity is given by O(n). Since this method requires the creation of the state space tree, its space complexity will also be exponential.

Solving an Example: Consider the problem with n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15. Here, we calculate the initial upper bound to be U = 10 + 10 + 12 = 32. Note that the 4th object cannot be included here, since that would exceed W. For the cost, we add 3/9 th of the final value, and hence the cost function is 38. Remember to negate the values after calculation before comparison.

After calculating the cost at each node, kill nodes that do not need exploring. Hence, the final state space tree will be as follows (Here, the number of the node denotes the order in which the state space tree was explored):



Note here that node 3 and node 5 have been killed after updating U at node 7. Also, node 6 is not explored further, since adding any more weight exceeds the threshold. At the end, only nodes 6 and 8 remain. Since the

value of U is less for node 8, we select this node. Hence the solution is {1, 1, 0, 1}, and we can see here that the total weight is exactly equal to the threshold value in this case.

Conclusion: We can solve 0/1 knapsack problem by using Dynamic Programming and Branch and Bound Approach.

Aim: Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen_s matrix.

Objective:

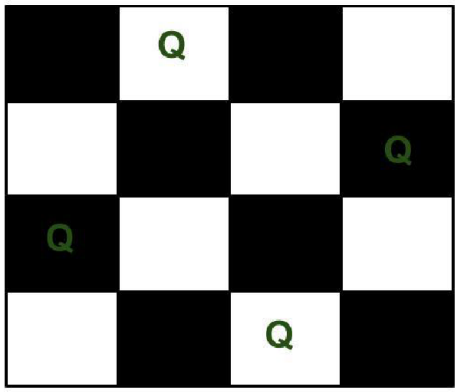
Students will learn

1. The Basic Concepts of N Queens Problem

2. N queens using Backtracking

Theory:

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, the following is a solution for the 4 Queen problems.



expected output is a binary matrix that has 1s for the blocks where queens are placed. For example, the following is the output matrix for the above 4 queen solution.

{ 0, 1, 0, 0}
{ 0, 0, 0, 1}
{ 1, 0, 0, 0}
{ 0, 0, 1, 0}

Backtracking Algorithm

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
the queen here leads to a 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 rows.
4) If all rows have been tried and nothing worked,
return false to trigger backtracking.
Time Complexity: O(N!)

Conclusion: We have studied N Queen Problem using Backtracking Approach.

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	