e-PG PATHSHALA- Computer Science
Design and Analysis of Algorithms
Module 9

Component-I (A) - Personal Details

Role	Name	Designation
Principal Investigator	Dr.T.V.Geetha	Senior Professor, Department of Computer Science &Engineering, Anna University, Chennai
Content Writer (CW)	Dr.S.Sridhar	Associate Professor Department of Information Science and Technology, Anna University, Chennai
Content Reviewer (CR)	Dr.K.S.Easwarakumar	Professor Department of Computer Science &Engineering, Anna University, Chennai
Language Editor (LE)		

Module 9

Component-I (B) Description of Module

Items	Description of Module
Subject Name	Computer Science
Paper Name	Design and Analysis of Algorithms
Module Name/Title	Exhaustive Searching and Combinatorial Optimization Problems
Module Id	CS/AlG/9
Pre-requisites	None
Objectives	To understand combinatorial optimization problems and exhaustive searching
Keywords	Algorithms, Problem Solving, Brute force, Exhaustive searching, Optimization Problems.

Module 9: Exhaustive Search and Combinatorial Optimization Problem

This module 9 focuses on the basics of the combinatorial optimization problems. The Learning objectives of this module are as follows:

- To introduce Combinatorial Optimization Problems and brute force approach
- To understand the concept of Exhaustive Searching
- To know about 15-Puzzle game
- To know about 8-Queen Problem
- To know about Knapsack Problem
- To understand assignment Problem

What is a brute force approach?

A brute force approach is a straight forward approach based on the problem statement and definitions of the concepts involved [2]. The general framework of brute force approach of brute force approach is as follows:

1. Generate all combinatorial structures that represent all feasible solutions. The combinatorial structures are sets, trees, graphs, permutations, and Catalan families.
2. Search for the best solution among the generated feasibility solutions.

The advantages of the brute force approach is as follows:

1. Wide applicability of brute force approaches. Brute force approach can be applied for all variety of problems.
2. Simplicity of the brute force approach.
3. Brute force approach can yield reasonable algorithms for all important problems.
4. Brute force approach can yield algorithms that can be used as a benchmark for comparing algorithms of other design approaches.

The disadvantage of brute force approaches are

1. Often brute force algorithms are inefficient.
2. There is not much constructive or creativity involved in brute force algorithms.

Optimization Problem and Exhaustive Searching

Exhaustive searching is a strategy usually employed for combinatorial problems. What is an optimization problem? An optimization problem has

1. Objective function: This is in the form of maximization or minimization of some constraint.
2. Predicate P that specifies the feasibility criteria
3. Solution space U with all possible solutions and extremum requirements
4. The aim is to find solution that satisfies the feasibility criteria.

Some of the examples of optimization problems that are discussed in module are 15-puzzle problem, 8-Queen problem, Knapsack problem and Assignment problem. Combinatorial optimization problem uses combinatorial structures such as sets, trees, graphs, permutations, and Catalan families.

Exhaustive search is a strategy for solving combinatorial optimization problems is to use exhaustive search. Exhaustive search involves these steps:

1. List all solutions of the problem
2. Often state space tree is used to represent all possible solutions.
3. Then the solution is searched in the state space tree that has least cost or optimal using a search technique.

This approach is using a brute force to search for finding solutions. The advantage is the guaranteed solution. But the disadvantage of this approach is a problem called “Combinatorial explosion” where the increase in input is associated with the rapid increase in output.

Let us discuss about applying brute force approach for some of the important problems now:

Solution Space and Unintelligent Search techniques:

The solution space is in the form of a graph. A sample solution space is shown in Fig. 2. Some of the important terminologies used in association with the solution space is given below:

1. Root: Root of a graph has no predecessor. In solution space, it is the initial configuration.
2. Solution Space: All the configurations in the form of a state space tree is called solution space. The target configuration is somewhere in the solution space.
3. Search techniques: Once a solution space is available, then the tree can be searched using search techniques like DFS and BFS.

Let us review about DFS and BFS now:

DFS Search

Depth First Search (DFS) is an unintelligent search that is used to search a graph. DFS uses a stack for traversing a graph. Initially, the root node is pushed onto a stack. Then, it is checked for goal node. If it is a goal node, then DFS reports success. Otherwise, its children is generated and pushed onto a stack. This procedure is repeated till stack is empty.

The procedure of DFS is given below:

```
Put the root node on a stack S;
    while (S is not empty) {
    remove a node from the S;
if (node = goal node) return success;
put all children of node onto the S;
 }
return failure;
```

The following example illustrates usage of DFS.

Example 1: Assume goal nodes are N and J. Show the DFS search of the following graph shown in Fig. 1 for finding the goal nodes?

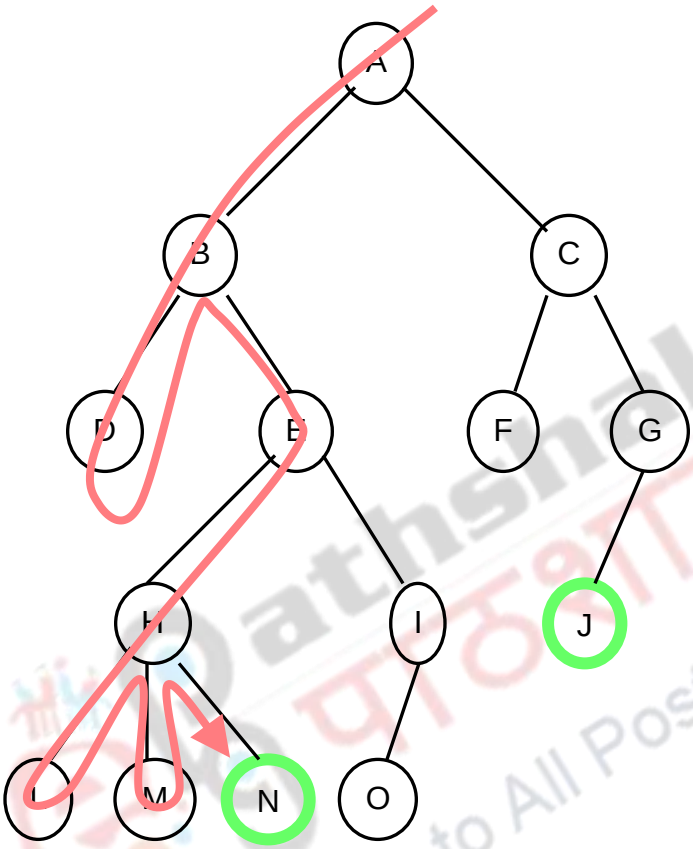

Fig 1: A sample Graph

Nodes that are explored in the order are ; A B D E L M N I O C F G J

BFS Search

Breadth First Search (BFS) is also an unintelligent search that is used to search a graph. BFS traverse a graph level-by-level. It uses a queue data structure for traversing a graph. Initially, the root node is added onto a queue. Then, it is checked for goal node. If it is a goal node, then BFS reports success. Otherwise, its children is generated and pushed onto a rear end of the queue Q. This procedure is repeated till Queue is empty.

The procedure of BFS is given below:

```
Put the root node on a queue Q;
while (Q is not empty) {
   remove a node from the queue Q;
   if (node = goal node) return success;
   put all children of node onto the queue Q;
}
return failure;
```

The following example illustrates usage of BFS.

Example 2: Assume goal nodes are N and J. Show the BFS search of the following graph shown in Fig. 2 for finding the goal nodes?

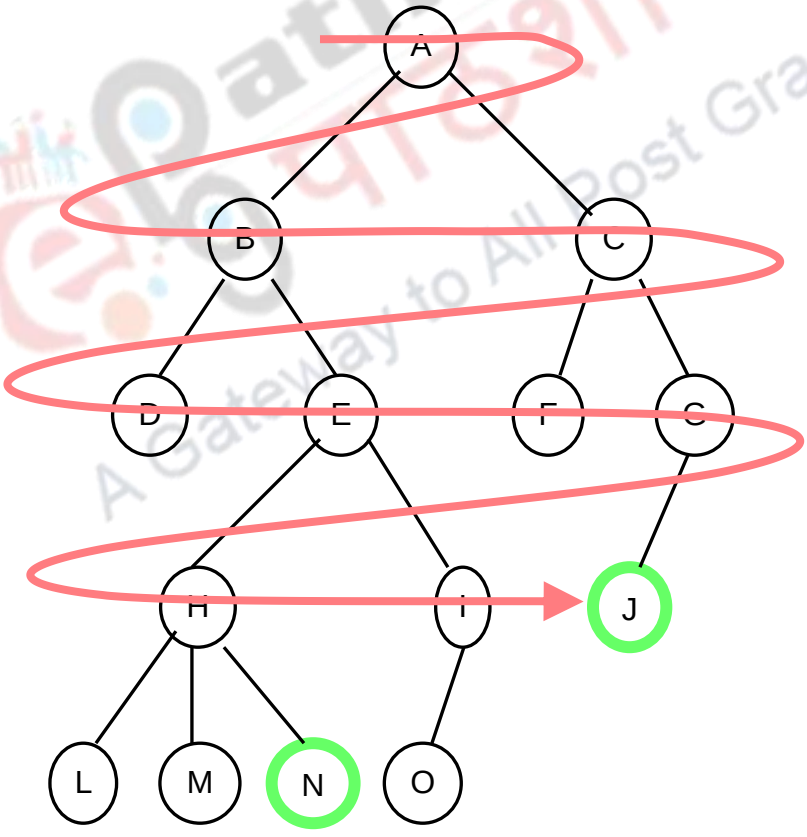

Fig 2: A sample Graph

The nodes that are explored in BFS order is : A B C D E F G H I J L M N O

15-Puzzle Problem

In 1878, Sam Lloyd designed a puzzle called 15-puzzle game. It is a game where a initial and target configuration is given. The game is about moving tile so that the target configuration is reached from the initial configuration. In other words, the objective of the game is to change initial state to goal state. The possible configurations of moving the up, down, right or left using the empty tile. So a node can have at most possible four moves. A sample initial and target configuration is given below in Fig. 3.

$$\begin{bmatrix} 1 & 5 & 12 & 13 \\ 2 & 6 & 11 & 14 \\ 3 & 7 & - & 15 \\ 4 & 8 & 9 & 10 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 5 & 12 & 13 \\ 2 & 6 & 11 & 14 \\ 3 & 7 & 15 & - \\ 4 & 8 & 9 & 10 \end{bmatrix}$$

Initial Configuration Target Configuration

Fig. 3. : Initial and Target configuration

A brute force algorithm can be written to generate the solution space. The repeated application of the legal moves, top, bottom, right and left results in a graph called state space or **solution space**. This is shown in Fig. 4.

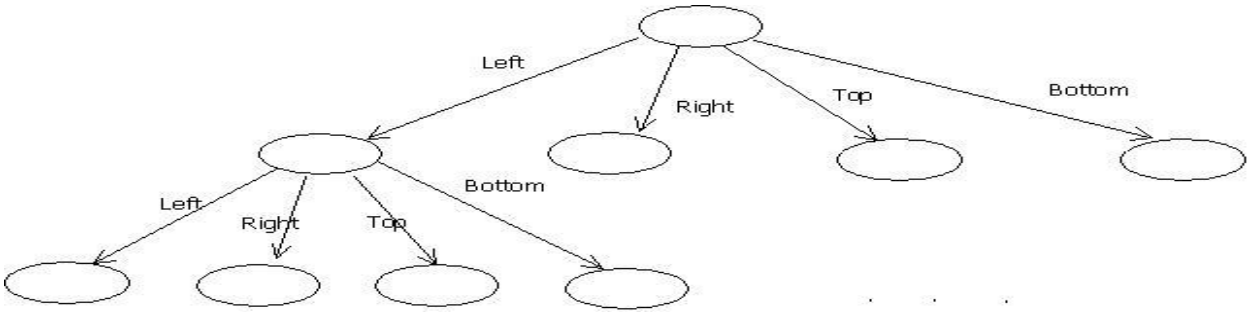

Fig. 4: **Portion of a Graph**

The goal state may be present somewhere in the solution space. Now, Exhaustive search technique is about finding the path from the starting node to the goal state. This may be done using DFS or BFS.

Formal Algorithm

The formal algorithm based on [1] for 15-puzzle is given as follows:

```
Algorithm 15puzzle (G, root, goal)
%% Input: State space tree with root and target goal state
%% Path from root to goal node
Begin
  for all nodes make visit = 0

 %% Let root be the starting node
      visited[root] = 1
      generate children w for root for all w do
      If (visited[w] = 0) and (w is not goal node) then
      call DFS(G,w)
     end if
  end for
End
```

Complexity Analysis

This brute force algorithm for 15-Puzzle game is inefficient and unrealistic. For 15-puzzle problem, there will be 16! ($\approx$ 20.9 X 10^{12}) different arrangements of tiles. Hence searching for the solution takes exponential time.

8-Queen Problem

The problem of eight queens is that of placing 8 queens in a non-attacking position. The problem of 8-queen problem is to generate a board configuration where the queens are in non-attacking positions.

The movement of queen is shown in Fig. 5. It can be observed that a queen can move in horizontal, vertical and diagonal ways.

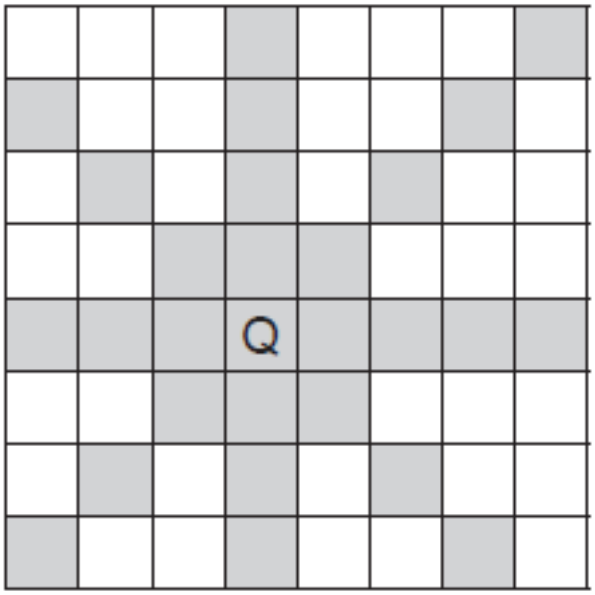

Fig. 5: The movement of a Queen

The brute force algorithm would be to try all possible combinations to check out the placement of the queens such that they are in a non-attacking position.

The informal algorithm is as follows:

1. Try all combinations
2. Check whether the queens are in attacking position
3. Repeat steps 1 and 2 until a valid configuration is possible.

The formal algorithm based on [1] is given as follows:

```
Algorithm 8queen(n)
%% Input: A 8 x 8 chess board and '8' queens
%% Output: Solution of 8-queen problem
Begin
 n = 8
      for i1 = 1 to n do for i2 = 1 to n do
          for i3 = 1 to n do for i4 = 1 to n do
               for i5 = 1 to n do for i6 = 1 to n do
                    for i7 = 1 to n do for i8 = 1 to n do
                      sol = [i1, i2, i3, i4, i5, i6, i7, i8]
                      %% check the solution with respect to constraints of
                      %% non-attacking queen
                        if sol is correct then print sol
                      End if
                    End for
               End for
          End for
      End for
    End for
  End for
 End for
End for
End.
```

Complexity Analysis:

Even though, the solution for 8-queen problem looks simple, in reality solution involves a combination of $\frac{64!}{56!}$ positions (56 = 64-8). This makes exhaustive search a difficult process. There are many solutions for the 8-queen problem.

Knapsack problem

Knapsack problem is one of the most popular algorithms that we often encounter in our daily life. It was designed by Denzig in 1950. A Knapsack problem has a knapsack of capacity K. There are n different items, each of which is associated with W_i and profit Pi (It is also called as value). The objective of knapsack problem is to load knapsack to maximize profit subjected to the capacity of knapsack.

Mathematical Formulation:

Mathematically, the problem can be stated as

$$Maximize \quad \sum_{i=1}^{n} p_i x_i$$

Subjected to the constraint that

$$\sum_{i=1}^{n} w_i x_i \le K$$

$$0 \le x_i \le 1 \text{ and } 1 \le i \le n$$

There are two types pf knapsack problem. It is easy to solve **fractional knapsack problem. This problem allows loading of knapsack with fractional items. This is possible if the items to be loaded are like** cloth, liquid, gold dust, etc. On the other hand, Integer knapsack problem, also known as 0/1 knapsack, is hard to solve. This problem is applicable for items which can either be loaded full or not at all. The items here are like electronic items, machineries or any items that can be broken.

Let us consider a scenario where there are three bottles of Salt with their weights and profits as shown in Fig. 6.

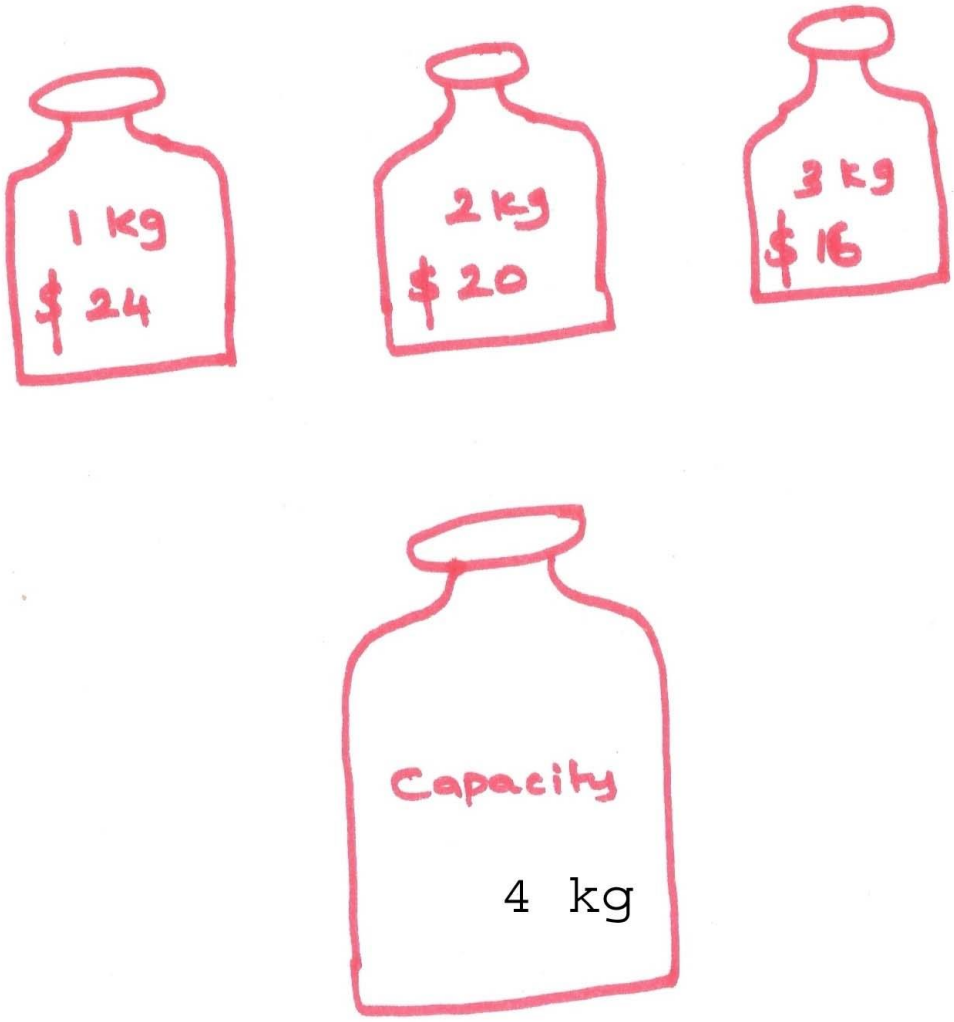

Fig.6 : Knapsack problem

The bigger bottle is knapsack. The capacities and profits are are shown in Fig. 6. So the question here is how to fill the knapsack effectively so that profit is higher subjected to the capacity of the knapsack.

Informal Algorithm:

1. Generate all possible sets. There will be 2^n possible subsets for 'n' items.
2. Generate a binary string of length
3. If the binary bit is 1, then include, otherwise exclude the item subjected to the constraints.
4. Compute the maximum profit and Weight for all combinations and report maximum profit.
5. Exit.

Formal Algorithm:

Formally, the knapsack problem based on [1] is given as follows:

```
Algorithm knapsack(K, w[1..n],p[1..n])
%% Input: Knapsack with capacity K , weights and profits
%% Output: Knapsack items and maximum weight and profit
Begin
 globalprofit = 0
choice = 0 index = 1
while (index <= 2^n ) do          %% for all 2^n subsets
  profit = 0
  weight = 0
    for i = 1 to n do             %% for all n items
      if (binary(index, i) == 0)  then    %% Item is excluded if binary bit is 0
        profit = profit + 0
        weight = weight +0                %% generate binary bit i for item index else
        profit = profit + p[i]            %% Update if Binary bit is 1
        weight = weight +w[i] End if
    End for
      if ((weight <= K) and (profit > globalprofit)) then   %% Check the constraints
      globalprofit = profit
      choice = i                          %% Note the combination
    End if
 index = index +1
 End while
End
```

This is illustrated in Example 3.

Example 3: Consider three items. Generate all the possibilities:

The possibilities are

φ

{1}

{2}

{3}

{1, 2}

{1, 3}

{2, 3}

{1,2,3}

Then all combinations are tried out subjected to the constraints. The profit is computed for every case and maximum profit is reported.

Complexity Analysis:

This involves generation of 2^n combinations. Hence, the complexity of the algorithm leads to a Ω (2^n) algorithm. The implication of NP-hard is that it would be difficult to solve the problem.

Assignment Problem

Assignment problem is to assign m persons to n jobs such that the cost is minimized. This problem can be solved using brute force approach. All possible combinations are tried out and cost is computed. Finally, the least cost assignment is reported.

Informal Algorithm

The informal algorithm is stated as below;

1. Generate all permutation of assignments
2. Compute the cost for all assignments
3. Choose the minimum cost assignment and report.
4. End.

Formal Algorithm

The formal algorithm based on [1] is given below:

Algorithm Assign(person[1..n],job[1..n])

%% Input; person and jobs

%% Output: Optimal assignment of person to job

Begin

permutate all legitimate assignments a_i
compute the assignment $[a_1, a_2, .., a_n]$ whose cost is minimum
assign as per minimum cost and return min_cost

End

The following Example 4 illustrates the application of this algorithm:

Example 4: A sample assignment Table in given in table 1.

Table 1: Sample assignment Table

	*J*1	J2	J3
person 1	3	4	5
person 2	2	3	7
person 3	8	9	2

Use assignment algorithm and find optimal cost?

There are 3 persons and 3 jobs. The question is how to assign the jobs to persons in an optimal manner based on the cost matrix given above. For example, if a random assignment is made like assign the job 1 to person 1 (C(1,1)), job 2 to person 2 (C(2,2)) and job 3 to person 3(C(3,3)), then the cost of the assignment would be

$< 1\ 2\ 3 > \ = \ 3 + 3 + 2 = 8$

For the above problem, all the possible cost assignments are given below in Table 2.

Table 2: Possible assignments

J1	J2	J3	Cost
1	2	3	3 + 3 + 2 = 8
1	3	2	3 + 7 + 9 = 19
2	1	3	4 + 2 + 2 = 8
2	3	1	4 + 7 + 8 = 19
3	1	2	5 + 2 + 9 = 16
3	2	1	5 + 3 + 8 = 16

It can be observed that the optimal order is < 1 2 3> and <2 1 3> as they are associated least cost.

Complexity Analysis

Again, the complexity analysis shows that the number of permutations are n!. Therefore, the complexity of the problem is O(n!).

Summary

In short, one can conclude as part of this module 9 that

- Brute force guarantee solutions but it is inefficient.
- It is difficult to solve combinatorial optimization problems.
- 15 Puzzle, 8-Queen, Knapsack and assignment problems are optimization problem that can be solved using brute force method.

References:

1. S.Sridhar – Design and Analysis of Algorithms, Oxford University Press, 2014.
2. A.Lvitin – Introduction to Design and Analysis of Algorithms, 2rd Edition, Pearson Education, 2014.
3. Cormen, T.H., C.E. Leiserson, and R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA 1992.
4. URL: http://www.hbmeyer.de/backtrack/achtdamen/eight.htm#up