UNIT-5

Back Tracking: Introduction, n-Queens problem, Sum of subsets, Hamiltonian cycle.

Branch and Bound: Introduction, Assignment problem, Travelling Salesman problem.

Introduction to Complexity classes: P and NP Problems, NP- Complete Problems.

Q) Define backtracking. List and explain various terminologies used in backtracking technique.

Many problems are difficult to solve algorithmically. Backtracking makes it possible to solve at least some large instances of difficult combinatorial problems.
Suppose we have to make a series of decisions among various choices, where

- We don’t have enough information to know what to choose
- Each decision leads to a new set of choices.
- Some sequence of choices (more than one choices) may be a solution to your problem.

The principal idea of **backtracking** is to construct solutions one component at a time and evaluate such partially constructed candidates as follows:

- If a partially constructed solution can be developed further without violating the problem’s constraints, it is done by taking the first remaining legitimate option for the next component.

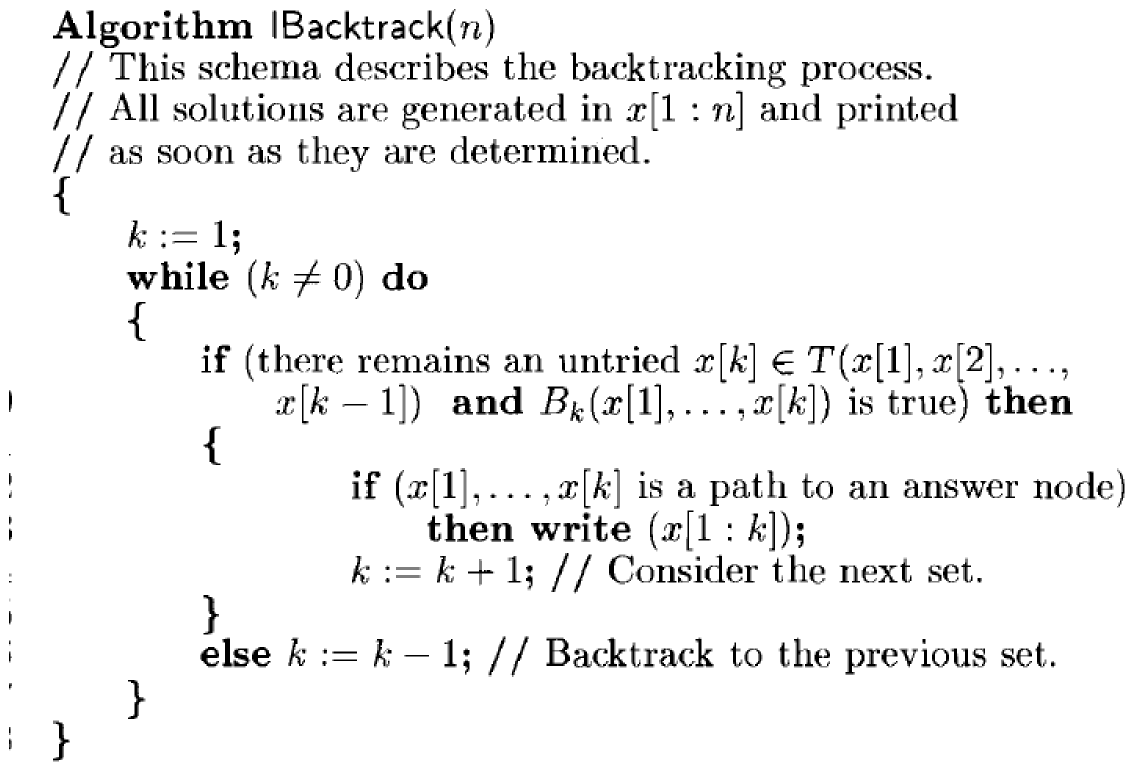
- If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

Applications of Backtracking:

- N Queens Problem
- Sum of subsets problem
- Graph coloring
- Hamiltonian cycles.

Terminology:
Problem state is each node in the depth-first search tree
State space is the set of all paths from root node to other nodes

Solution states are the problem states **s** for which the path from the root node to **s**
Answer states are that solution states s for which the path from root node to s defines a tuple that is a member of the set of solutions
State space tree is the tree organization of the solution space
Live node is a generated node for which all of the children have not been generated yet.
E-node is a live node whose children are currently being generated or explored
Dead node is a generated node that is not to be expanded any further



```
Algorithm IBacktrack(n)
// This schema describes the backtracking process.
// All solutions are generated in x[1 : n] and printed
// as soon as they are determined.
{
    k := 1;
    while (k ≠ 0) do
    {
        if (there remains an untried x[k] ∈ T(x[1], x[2], ...,
            x[k − 1]) and B_k(x[1], ..., x[k]) is true) then
        {
                if (x[1], ..., x[k] is a path to an answer node)
                    then write (x[1 : k]);
                k := k + 1; // Consider the next set.
        }
        else k := k − 1; // Backtrack to the previous set.
    }
}
```

Alg. General iterative backtracking method

Q) Explain N-Queens problem using backtracking.

N-Queens Problem:

A classic combinational problem is to place n queens on a n*n chess board so that no two attack, i.e; no two queens are on the same row, column or diagonal.

If we take n=4 then the problem is called 4 queens problem.

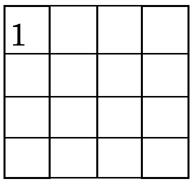
If we take n=8 then the problem is called as 8 queens problem.

4-Queens Problem:

Consider a 4*4 chessboard. Let there are 4 queens. The objective is place there 4 queens on 4*4 chessboard in such a way that no two queens should be placed in the same row, same column or diagonal position.

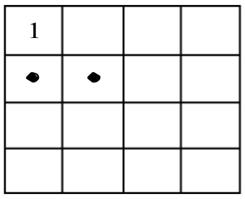
The explicit constraints are 4 queens are to be placed on 4*4 chessboards in 44 ways.

The implicit constraints are no two queens are in the same row column or diagonal. Let{x1, x2, x3, x4} be the solution vector where x1 column on which the queen i is placed. First queen is placed in first row and first column.

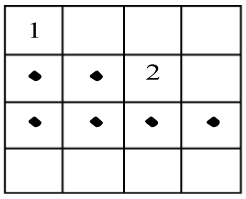


(a)

The second queen should not be in 1st row and 2nd column. It should be placed in 2nd row and in 2nd, 3rd or 4th column. So we place in second column, both will be in same diagonal, so place it in third column.

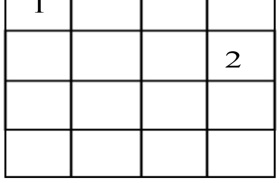


(b)

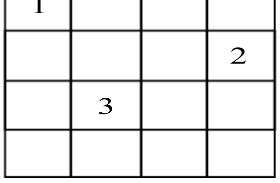


(c)

We are unable to place queen 3 in third row, so go back to queen 2 and place it somewhere else

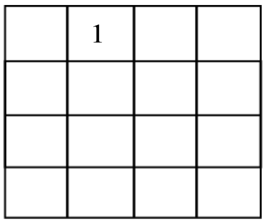


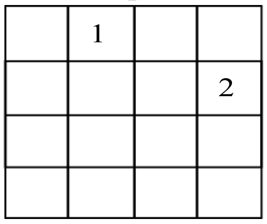
(d)



(e)

Now the fourth queen should be placed in 4th row and 3rd column but there will be a diagonal attack from queen 3. So go back, remove queen 3 and place it in the next column. But it is not possible, so move back to queen 2 and remove it to next column but it is not possible. So go back to





queen 1 and move it to next column.

(f) (g)

	1		
			2
3			

(h)

1			
			2
3			
		4	

(i)

Fig: Example of Backtrack solution to the 4-queens problem

Hence the solution of to 4-queens's problem is x1=2, x2=4, x3=1, x4=3, i.,e first queen is placed in 2nd column, second queen is placed in 4th column and third queen is placed in first column and fourth queen is placed in third column.

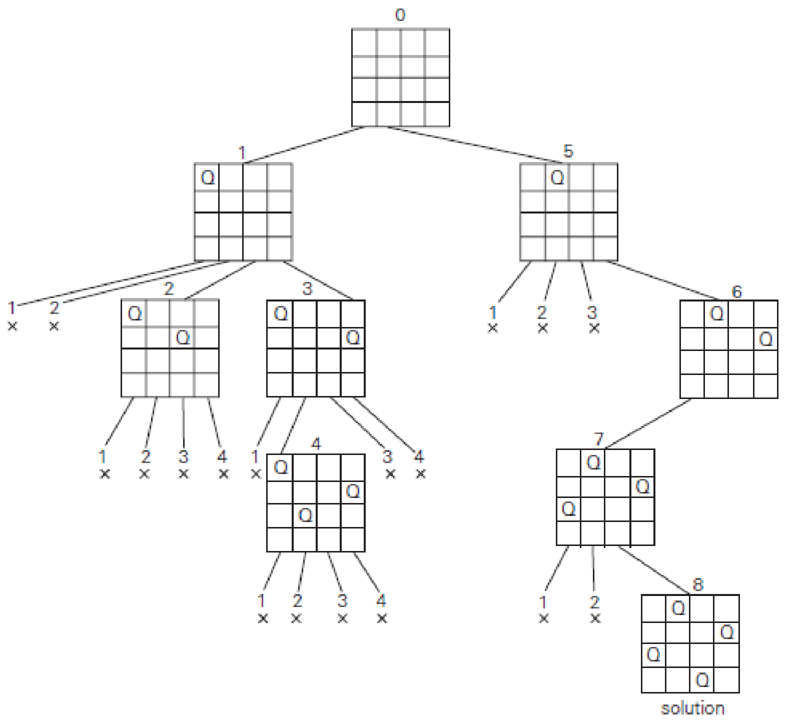


Fig. State-space tree of solving the four-queens problem by backtracking.

× denotes an unsuccessful attempt to place a queen in the indicated

column. The numbers above the nodes indicate the order in which the

nodes are generated.

```
Algorithm NQueens($k, n$)
// Using backtracking, this procedure prints all
// possible placements of $n$ queens on an $n \times n$
// chessboard so that they are nonattacking.
{
    for $i := 1$ to $n$ do
    {
        if Place($k, i$) then
        {
            $x[k] := i$;
            if ($k = n$) then write ($x[1 : n]$);
            else NQueens($k + 1, n$);
        }
    }
}

Algorithm Place($k, i$)
// Returns true if a queen can be placed in $k$th row and
// $i$th column. Otherwise it returns false. $x[\ ]$ is a
// global array whose first ($k - 1$) values have been set.
// Abs($r$) returns the absolute value of $r$.
{
    for $j := 1$ to $k - 1$ do
        if (($x[j] = i$) // Two in the same column
            or (Abs($x[j] - i$) = Abs($j - k$)))
                // or in the same diagonal
            then return false;
    return true;
}
```

8-queens problem

A classic combinatorial problem is to place 8 queens on a 8*8 chess board so that no two attack, i.,e no two queens are to the same row, column or diagonal.

Now, we will solve 8 queens problem by using similar procedure adapted for 4 queens problem. The algorithm of 8 queens problem can be obtained by placing n=8, in N queens algorithm.

If two queens are placed at positions (i,j) and (k,l). They are on the same diagonal only if

i-j=k-l ……………….(1) or

i+j=k+l................(2).

From (1) and (2) implies j-l=i-k and j-l=k-i

Two queens lie on the same diagonal iff |j-l|=|i-k|

The solution of 8 queens problem can be obtained similar to the solution of 4 queens. problem.X1=3, X2=6, X3=2, X4=7, X5=1, X6=4, X7=8, X8=5,
The solution can be shown as

		1					
					2		
	3						
						4	
5							
			6				
							7
				8			

Q) Explain inbrief about sum of subsets problem.

Sum of Subsets Problem

Subset sum problem is the problem of finding a subset such that the sum of elements equal a given number. The backtracking approach generates all permutations in the worst case but in general, performs better than the recursive approach towards subset sum problem.

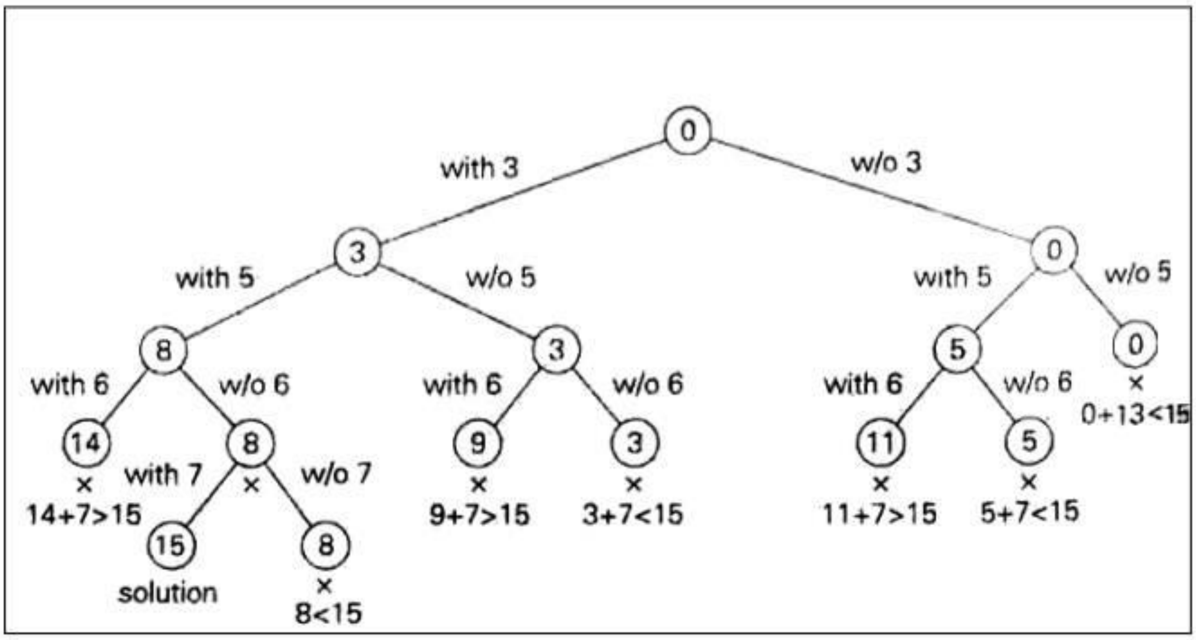
A subset A of n positive integers and a value sum(d) is given, find whether or not there exists any subset of the given set, the sum of whose elements is equal to the given value of sum.

Steps:

1. Start with an empty set
2. Add the next element from the list to the set
3. If the subset is having sum M, then stop with that subset as solution.
4. If the subset is not feasible or if we have reached the end of the set, then backtrack through the subset until we find the most suitable value.
5. If the subset is feasible (sum of subset < d) then go to step 2.

6. If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution.

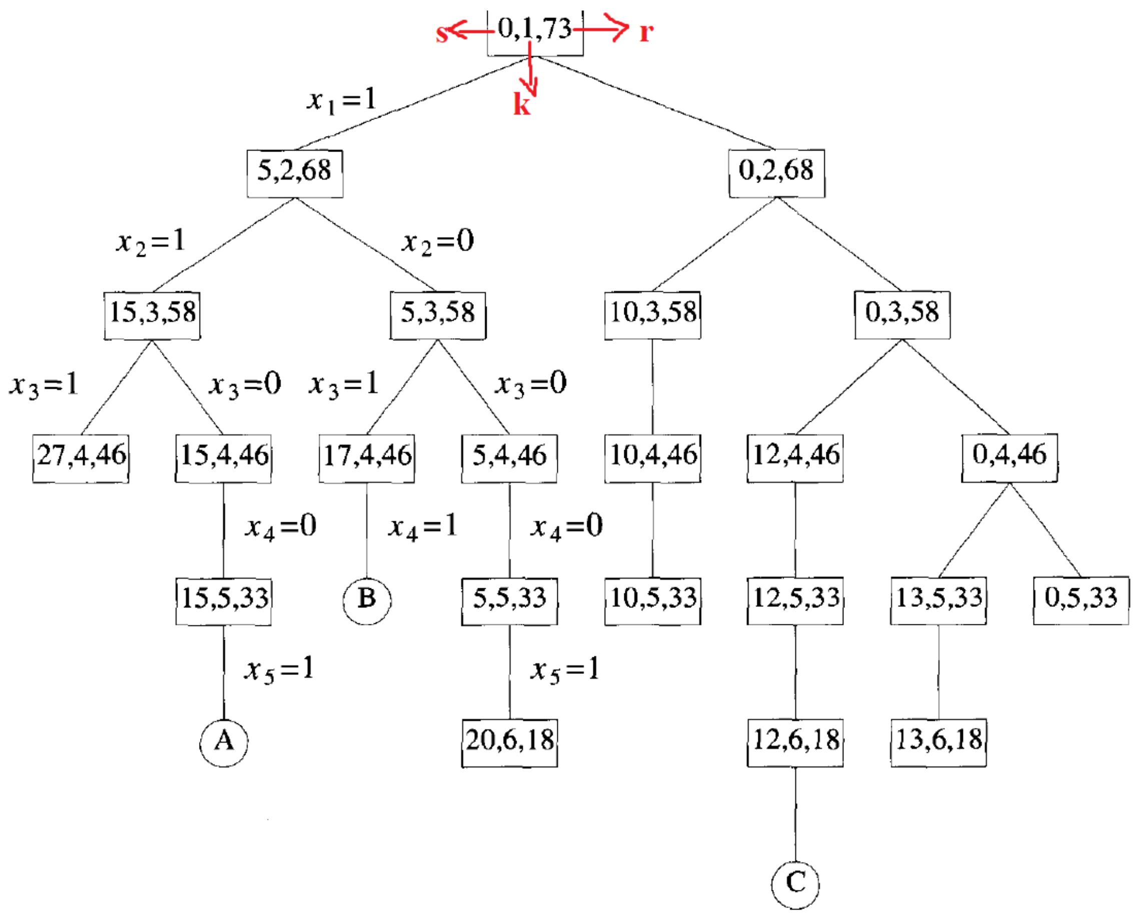
Example: S = {3,5,6,7} and d = 15, Find the sum of subsets by using backtracking



Solution {3,5,7}

Example 2:

S={5,10,12,13,15,18}, required sum = 30



```
Algorithm SumOfSub($s, k, r$)
// Find all subsets of $w[1:n]$ that sum to $m$. The values of $x[j]$,
// $1 \le j < k$, have already been determined. $s = \sum_{j=1}^{k-1} w[j] * x[j]$
// and $r = \sum_{j=k}^{n} w[j]$. The $w[j]$'s are in nondecreasing order.
// It is assumed that $w[1] \le m$ and $\sum_{i=1}^{n} w[i] \ge m$.
{
    // Generate left child. Note: $s + w[k] \le m$ since $B_{k-1}$ is true.
    $x[k] := 1$;
    if $(s + w[k] = m)$ then write $(x[1:k])$; // Subset found
        // There is no recursive call here as $w[j] > 0,\ 1 \le j \le n$.
    else  if $(s + w[k] + w[k+1] \le m)$
            then SumOfSub($s + w[k], k+1, r - w[k]$);
    // Generate right child and evaluate $B_k$.
    if $((s + r - w[k] \ge m)$ and $(s + w[k+1] \le m))$ then
    {
        $x[k] := 0$;
        SumOfSub($s, k+1, r - w[k]$);
    }
}
```

Q) Briefly explain hamiltonian Circuits Problem with suitable example.

A Hamiltonian circuit or tour of a graph is a path that starts at a given vertex, visits each vertex in the graph exactly once, and ends at the starting vertex.

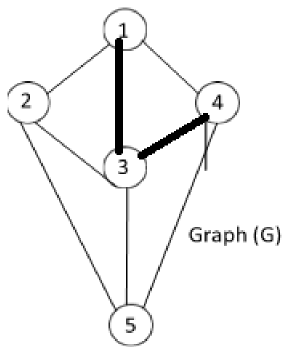
We use the Depth-First Search algorithm to traverse the graph until all the vertices have been visited.

We traverse the graph starting from a vertex (arbitrary vertex chosen as starting vertex) and at any point during the traversal we get stuck (i.e., all the neighbor vertices have been visited), we backtrack to find other paths (i.e., to visit another unvisited vertex).

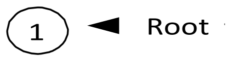
If we successfully reach back to the starting vertex after visiting all the nodes, it means the graph has a Hamiltonian cycle otherwise not.

We mark each vertex as visited so that we don't traverse them more than once.

Example: Find the Hamiltonian circuits of a given graph using Backtracking.

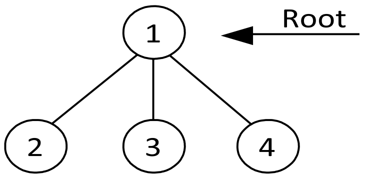


Initially we start out search with vertex '1' the vertex '1' becomes the root of our implicit tree.



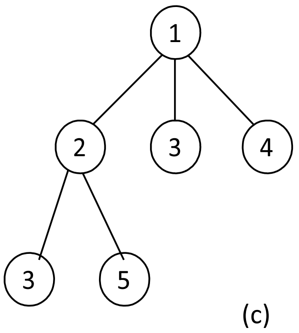
(a)

Next we choose vertex '2' adjacent to '1', as it comes first in numerical order (2, 3, 4).



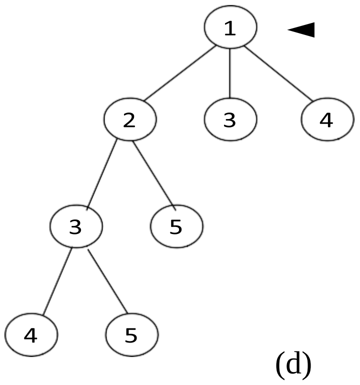
(b)

Next vertex '3' is selected which is adjacent to '2' and which comes first in numerical order (3,5).



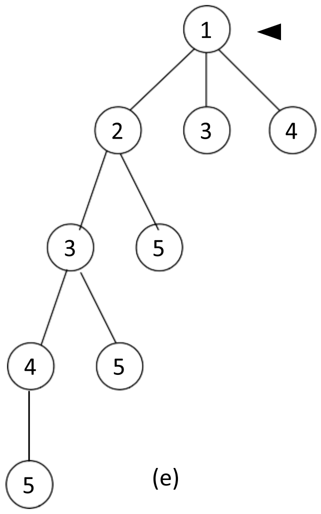
(c)

Next we select vertex '4' adjacent to '3' which comes first in numerical order (4, 5).



(d)

Next vertex '5' is selected. If we choose vertex '1' then we do not get the Hamiltonian cycle.

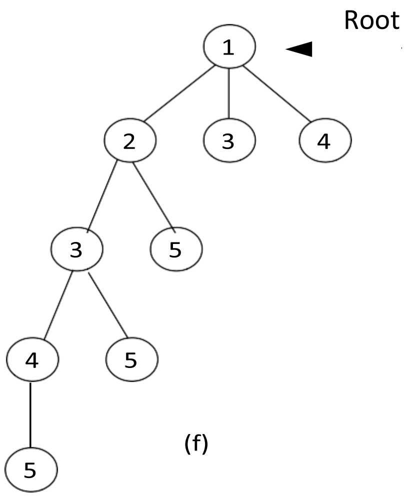


(e)

The vertex adjacent to 5 is 2, 3, 4 but they are already visited. Thus, we get the dead end. So, we backtrack one step and remove the vertex '5' from our partial solution.
The vertex adjacent to '4' are 5,3,1 from which vertex '5' has already been checked and we are left with vertex '1' but by choosing vertex '1' we do

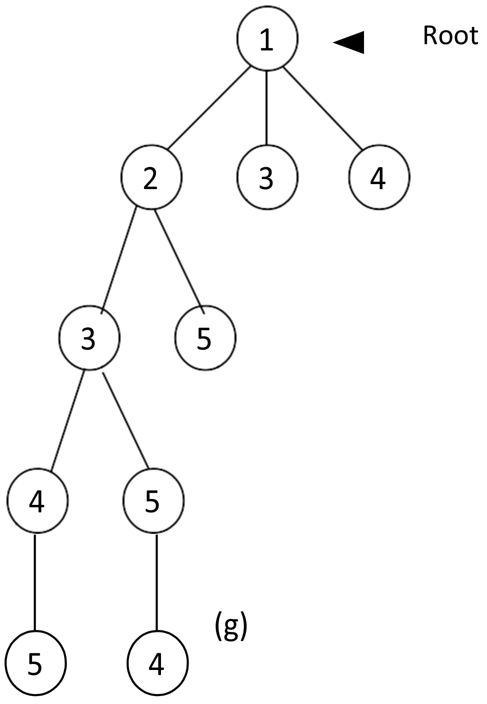
not get the Hamiltonian cycle. So, we again backtrack one step.

Hence we select the vertex ‘5’ adjacent to ‘3’.



(f)

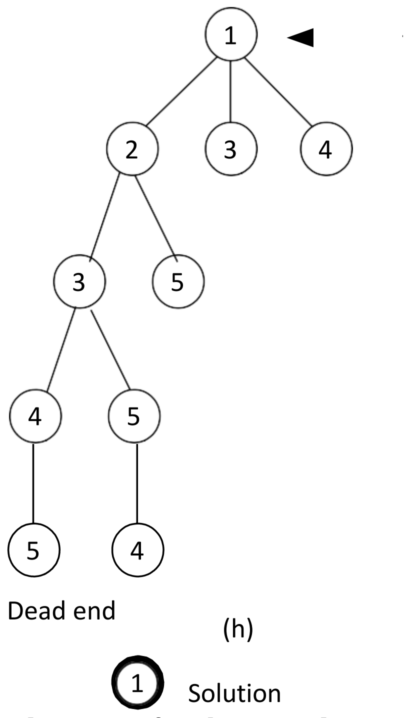
The vertex adjacent to ‘5’ are (2,3,4) so vertex 4 is selected.



(g)

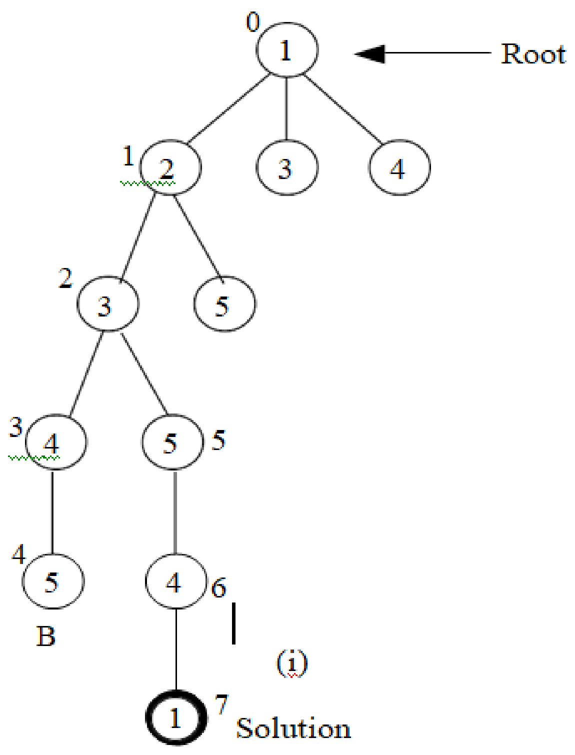
Dead end

The vertex adjacent to ‘4’ are (1, 3, 5) so vertex ‘1’ is selected. Hence we get the Hamiltoniancycle as all the vertex other than the start vertex ‘1’ is visited only once, 1- 2- 3- 5- 4- 1.



(h)

The final implicit tree for the Hamiltonian circuit is shown below. The number above eachnode indicates the order in which these nodes are visited.



(i)

Algorithm:

```
Algorithm Hamiltonian($k$)
// This algorithm uses the recursive formulation of
// backtracking to find all the Hamiltonian cycles
// of a graph. The graph is stored as an adjacency
// matrix $G[1:n, 1:n]$. All cycles begin at node 1.
{
    repeat
    { // Generate values for $x[k]$.
        NextValue($k$); // Assign a legal next value to $x[k]$.
        if ($x[k] = 0$) then return;
        if ($k = n$) then write ($x[1:n]$);
        else Hamiltonian($k+1$);
    } until (false);
}
```

```
Algorithm NextValue($k$)
// $x[1:k-1]$ is a path of $k-1$ distinct vertices. If $x[k] = 0$, then
// no vertex has as yet been assigned to $x[k]$. After execution,
// $x[k]$ is assigned to the next highest numbered vertex which
// does not already appear in $x[1:k-1]$ and is connected by
// an edge to $x[k-1]$. Otherwise $x[k] = 0$. If $k = n$, then
// in addition $x[k]$ is connected to $x[1]$.
{
    repeat
    {
        $x[k] := (x[k] + 1)$ mod $(n + 1)$; // Next vertex.
        if ($x[k] = 0$) then return;
        if ($G[x[k-1], x[k]] \neq 0$) then
        { // Is there an edge?
            for $j := 1$ to $k-1$ do if ($x[j] = x[k]$) then break;
                            // Check for distinctness.
            if ($j = k$) then // If true, then the vertex is distinct.
                if (($k < n$) or (($k = n$) and $G[x[n], x[1]] \neq 0$))
                    then return;
        }
    } until (false);
}
```

Q) Briefly explain Branch and Bound technique.

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

Branch and bound is an algorithmic technique to find the **optimal solution** by keeping the best solution found so far. If a partial solution cannot improve on the best, it is abandoned.

Branch and Bound differs from backtracking in two important points:

- It has a branching function, which can be a depth first search, breadth first search or based on bounding function.
- It has a bounding function, which goes far beyond the feasibility test as a mean to prune efficiently the search tree.

Branch and Bound refers to all state space search methods in which all children of the E-node are generated before any other live node becomes the E-node.

- Live node is a node that has been generated but whose children have not yet been generated.
- E-node is a live node whose children are currently being explored. In other words, an E-node is a node currently being expanded.
- Dead node is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded.
- Branch-an-bound refers to all state space search methods in which all children of an E-node are generated before any other live node can become the E-node.

Q) Explain about Assignment problem with suitable example using branch and bound method.

Assigning n people to n jobs so that the total cost of the assignment is as small as possible. Select one element in each row of the matrix so that no two selected elements are in the same column and their sum is the smallest possible.

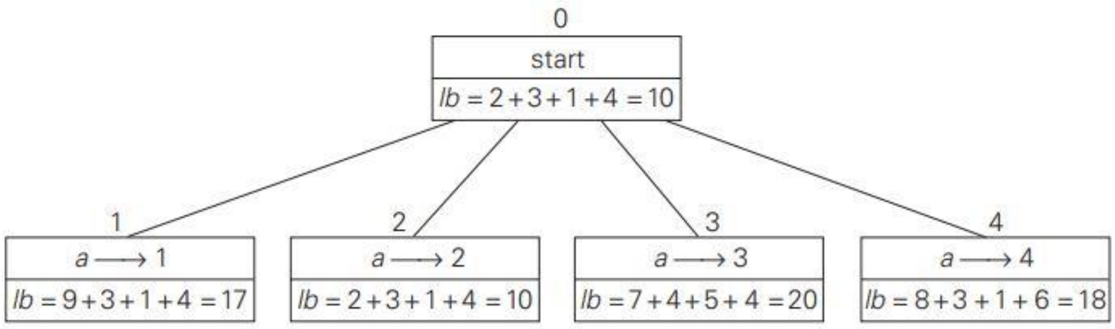
Let there be N workers and N jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the job assignment. It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.

$$\begin{array}{cccccc} & \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} & \\ & 9 & 2 & 7 & 8 & \text{person } a \\ C = & 6 & 4 & 3 & 7 & \text{person } b \\ & 5 & 8 & 1 & 8 & \text{person } c \\ & 7 & 6 & 9 & 4 & \text{person } d \end{array}$$

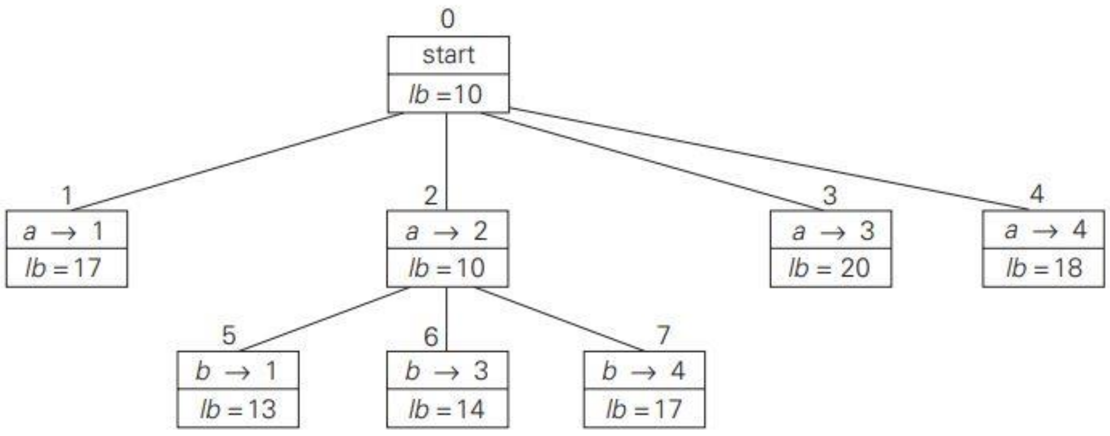
Lb= the sum of the smallest elements in each of the matrix’s rows.

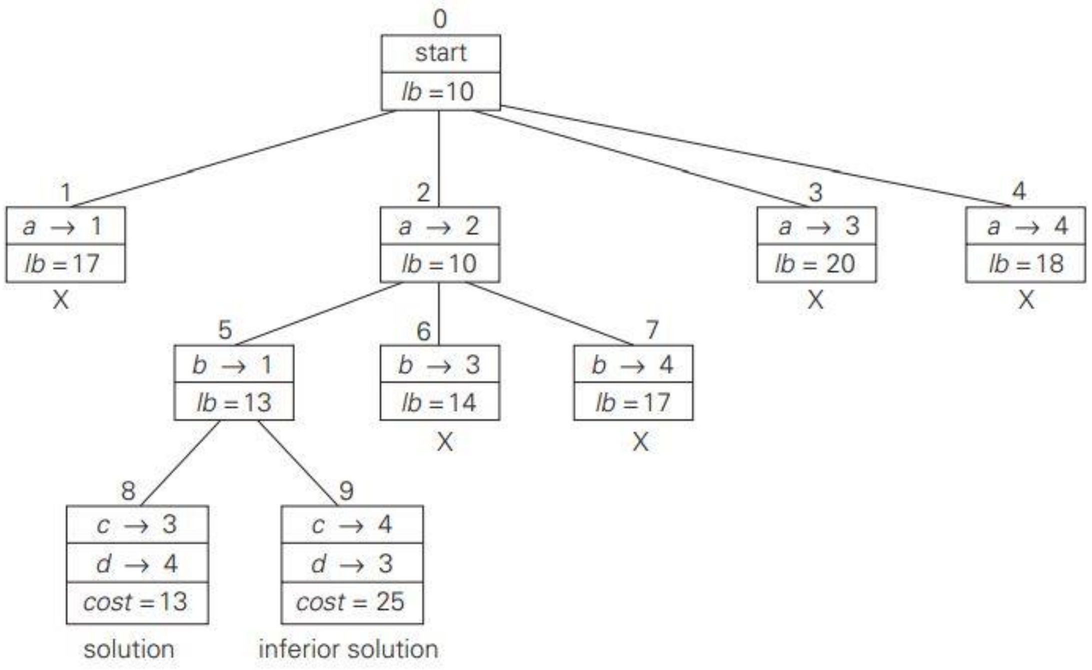
For the instance here, this sum is 2 + 3 + 1 + 4 = 10.

The lower-bound value for the root, denoted lb, is 10. The nodes on the first level of the tree correspond to selections of an element in the first row of the matrix, i.e., a job for person a



The most promising of them is node 2 because it has the smallest lower bound value.We branch out from that node first by considering the three different ways of selecting an element from the second row and not in the second column the three different jobs that can be assigned to person b.





Q) Explain about Traveling Sale Person problem with suitable example.

Travelling Salesman Problem states-

- A salesman has to visit every city exactly once.
- He has to come back to the city from where he starts his journey.
- What is the shortest possible route that the salesman must follow to complete his tour?

Example: Find the LC branch and bound solution for the traveling sale person problem whose cost matrix is as follows

$$\begin{pmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{pmatrix}$$

Step 1: Find the reduced cost matrix.

Apply row reduction method:

Deduct 10 (which is the minimum) from all values in the 1st row.

Deduct 2 (which is the minimum) from all values in the 2nd row.

Deduct 2 (which is the minimum) from all values in the 3rd row.

Deduct 3 (which is the minimum) from all values in the 4th row.

Deduct 4 (which is the minimum) from all values in the 5th row.

The resulting row wise reduced cost matrix =

$$\begin{pmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{pmatrix}$$

Row wise reduction sum = 10 + 2 + 2 + 3 + 4 = 21

Apply column reduction method:

Deduct 1 (which is the minimum) from all values in the 1st column.

Deduct 3 (which is the minimum) from all values in the 2nd column.

The resulting row wise reduced cost matrix =

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix} \longrightarrow \boxed{A}$$

Initial reduction cost = row red cost + col red cost

= 21+4

= 25

$1 \longrightarrow 2$ $\quad \underset{\text{row}}{\overset{\downarrow}{1}} \quad \underset{\text{column}}{\overset{\downarrow}{2}}$

$\underbrace{\text{row } 1,\ \text{col } 2,\ (2,1)}$ $\to \underset{\text{must be set to } \infty \text{ in A matrix}}{\infty}$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

check whether each row and each col is having atleast one zero.

If not having atleast one zero we've perform reduction process.

reduced cost = row red cost + col red cost

= 0+0=0

cost(1,2) = initial reduced cost + reduced cost of (1,2) + $\underset{\text{reduced matrix}}{A[1,2]}$ value from ↑

= 25+0+10

= 35

cost(1,3)

$1 \to 3$ $\quad \underset{\text{row}}{\overset{\downarrow}{1}} \quad \underset{\text{col}}{\overset{\downarrow}{3}}$

row 1, col 3, (3,1) $\to \infty$

$$\begin{array}{c} \\ 1 \\ \\ 4 \\ 0 \\ \\ \end{array} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 19 & \infty \end{bmatrix}$$

11

red cost = row red cost + col red cost

= 0 + 11

= 11

$\text{cost}(1,3) = 25 + 11 + A[1,3]$

$= 25 + 11 + 17$

$= 53$

$1 \to 4$, row 1, col 4, $(4,1) \to \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \longrightarrow B$$

0

red cost = row red cost + col red cost

$= 0 + 0$

$= 0$

$\text{cost}(1,4) = 25 + 0 + A[1,4]$

$= 25 + 0 + 0$

$= 25$

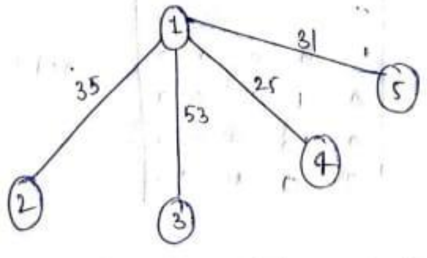
$1 \to 5$ row 1, col 5, $(5,1) \to \infty$

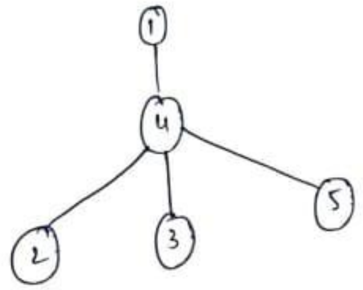
$$\begin{array}{c|c} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} & \begin{matrix} \\ 2 \\ \\ 3 \\ \\ \end{matrix} \end{array} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

red cost = 5

$\text{cost}(1,5) = 25 + 5 + 1$

$= 31$





(4,2) 4th row, 2nd col, (2,1) → ∞

4→2

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix} \longrightarrow \boxed{C}$$

red cost = 0 + 0
= $\underline{0}$

$\underline{(1,4) \rightarrow 2}$

$$\text{cost}(4,2) = \underset{\substack{\downarrow \\ \text{red cost}}}{0} + \underset{\substack{\downarrow \\ \text{cost of } (1,4)}}{25} + \overset{B}{B}[4,2]$$

= 25 + 3 = 28

(4,3) 4th row, 3rd col, (3,1) → ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix} 2 \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix}$$
11

$$\Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

red cost = 13

$$\begin{array}{r} 1 \\ 25 \\ 12 \\ 13 \\ \hline 50 \end{array}$$

cost(4,3) = 13 + 25 + B[4,3]
= 13 + 25 + 12 = 50

(4,5) $\to$ 4th row, 5th col, $(4,1) \to \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} \\ 11 \\ \\ \\ \\ \end{matrix} \to \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & 0 & \infty & \infty \end{bmatrix}$$

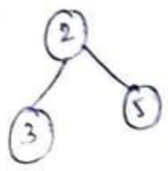
red cost = 11

cost(4,5) = 11 + 25 + B[4,5]

= 36 + 0

= 36

<u>(1,4,2)</u>



<u>(2,3)</u> 2nd row, 3rd col, $(3,1) \to \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} \\ \\ 2 \\ \\ 11 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

2

$$\Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

red cost = <u>13</u>

cost(2,3) = 13 + 28 + c[2,3]

= 13 + 28 + 11 = 52

(2,5) 2nd row, 5th col ,(5,1) → ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix} \quad \begin{matrix} \\ \longrightarrow 0 \\ \\ \\ \\ \end{matrix}$$

red cost = 0

$$\text{cost}(2,5) = 0 + 28 + C[2,5]$$

$$= 0 + 28 + 0$$

$$= 28$$

(5,3) 5th row, 3rd col (3,1) → ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

red cost = 0

$$\text{cost}(5,3) = 0 + 28 + D[5,3]$$

$$= 0 + 28 + 0$$

$$= \underline{28}$$

1 → 4 → 2 → 5 → 3 → 1 with least cost as 28