Chapter-6
Backtracking

6.1 Background

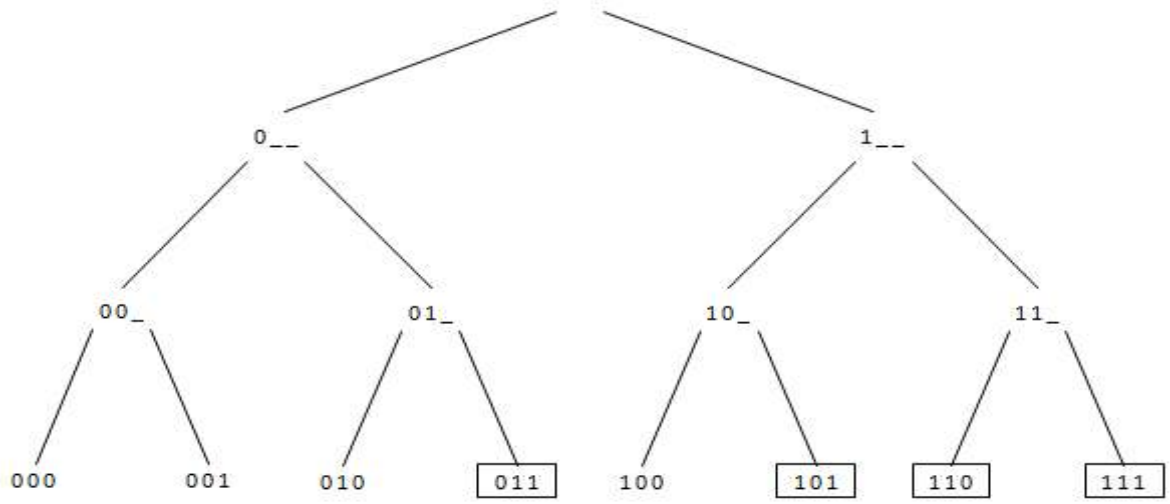
Suppose, if you have to make a series of decisions, among various choices, where you don't have enough information to know what to choose. Each decision leads to a new set of choices. Some sequence of choices may be solution to your problem. Backtracking is a methodical way of trying out various sequences of decisions, until you find one that "works".

- Backtracking is used to solve problems in which a sequence of objects is chosen from a specified set so that the sequence satisfies some criterion.
- We call a node *non promising* if when visiting the node we determine that it cannot possibly lead to a solution. Otherwise, we call it *promising.*
- In summary, backtracking consists of
- Doing a depth-first search of a state space tree,
- Checking whether each node is promising, and, if it is non promising, backtracking to the node's parent.
- This is called *pruning* the state space tree, and the sub tree consisting of the visited nodes is called the *pruned state space tree*.
- Definition: A general algorithm for finding solution(s) to a computational problem by trying partial solutions and then abandoning them ("backtracking") if they are not suitable.

Back tracking example problem

Find out all 3-bit binary numbers for which the sum of the 1's is greater than or equal to 2.
The only way to solve this problem is to check all the possibilities:

(000, 001, 010... 111)



6.2 General Back tracking algorithm

Step 1: We build a partial solution v = (a (1), a (2)... a (k)), extend it and test it.
Step 2: If the partial solution is still a candidate solution
Proceed.
Else
Delete a (k) and try another possible choice for a (k).
Step 3: If possible choices of a (k) are exhausted, backtrack and try the next choice for a (k-1)
In case of Greedy method and Dynamic programming techniques, we will use Brute Force approach. It means we will evaluate all possible solutions, among which we select one solution as optimal solution. In back tracking technique, we will get same optimal solution with less number of steps. The main advantage of back tracking is, if a partial solution (X1, X2, X3 ….Xi) can't lead to optimal solution then (Xi+1 …...X n) solution may be ignored entirely.

Explicit constrains: These are the rules which restrict each Xi to take on values only from a given set.

Implicit constraints: These are the rules which determine which of the tuples in the solution space satisfies the criterion functions.
Terminology used in this method.
1. *Criterion function*: It is a function P(X1, X2… X n) which needs to be maximized or minimized for a given problem.

2. Solution Space: All tuples that satisfy the explicit constraints define a possible solution space for a particular instance 'I' of the problem

3. Problem state: Each node in the tree organization defines a problem state.

4. Solution States: These are those problem states S for which the path form the root to S define a tuple in the solution space.

5. State space tree: If we represent solution space in the form of a tree then the tree is referred as the state space tree.

6. Answer States: These solution states S for which the path from the root to S defines a tuple which is a member of the set of solution (i.e. it satisfies the implicit constraints) of the problem.

7. Live node: A node which has been generated and all of whose children have not yet been generated is live node.

8. E-node: The live nodes whose children are currently being generated is called E-node (node being expanded)

9. Dead node: It is a generated node that is either not to be expanded further or one for which all of its children has been generated.
10. Bounding function: It will be used to kill live nodes without generating all their children

11. Branch and bound: It is a method in which E-node remains E-node until it is dead.

Applications of Backtracking

producing all permutations of a set of values

parsing languages

Games: anagrams, crosswords, word jumbles, 8 queens

Combinatory and logic programming

Example Applications

i. 4 queen's problem

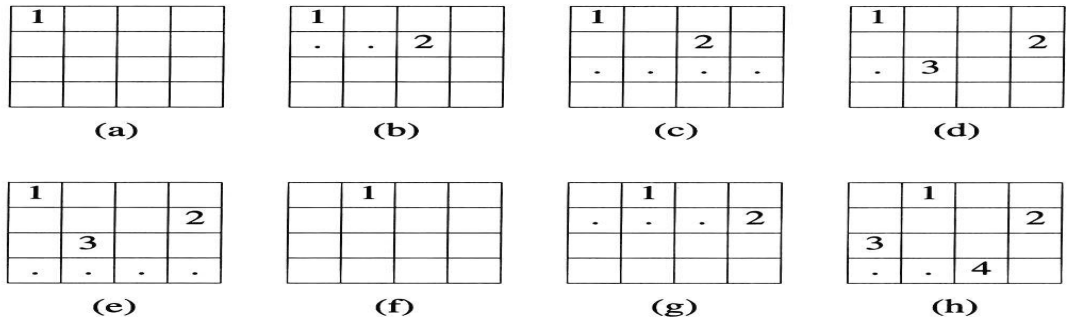
ii. 8 queen's problem

iii. N queen's problem

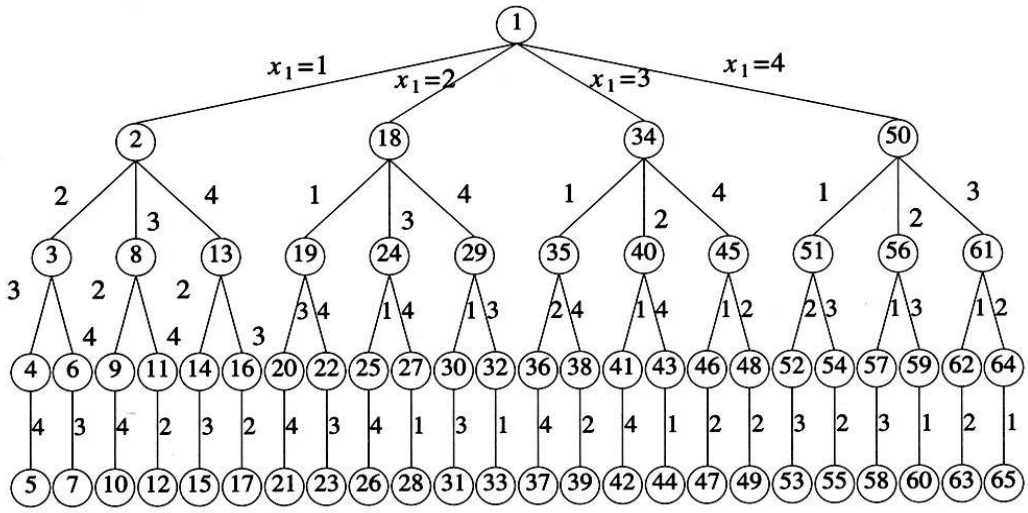
iv. Sum of subsets problem.

6.3 Queens problem

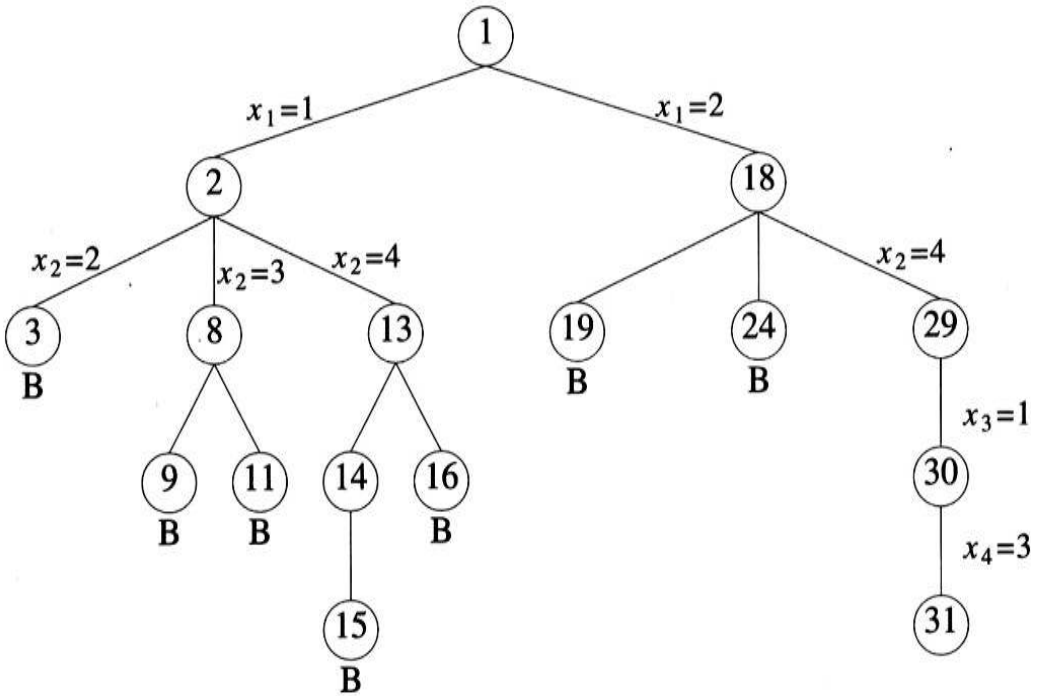
The objective of this problem is to place 4 queens on 4X4 chess board in such a way that no two queens should placed in the same row, same column or diagonal position.
Explicit constraint: 44 ways
Implicit constraints: No two queens should in same row, same column or diagonal position.



Searching the solution space for this problem by using a tree organization.



A portion of tree that is generated during backtracking is



Explanation

i)If (x1….xi) is the path to the current E-node, a bounding function has the criterion that (x1...xi+1) represents a chessboard configuration, in which no queens are attacking.

ii)A node that gets killed as a result of the bounding function has a B under it.

iii)We start with root node as the only live node. The path is ( ); we generate a child node 2.

iv)The path is (1).This corresponds to placing queen 1 on column 1.

v)Node 2 becomes the E node. Node 3 is generated and immediately killed. (Because x1=1, x2=2).

vi)As node 3 is killed, nodes 4,5,6,7 need not be generated.

vii)Node 8 is generated, and the path is (1, 3).

viii)Node 8 gets killed as all its children represent board configurations that cannot lead to answer.

ix)We backtrack to node 2 and generate another child node 13.

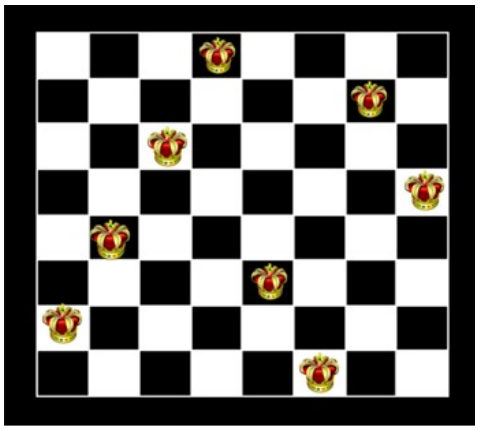
x)But the path (1, 4) cannot lead to answer nodes.

So, we backtrack to 1 and generate the path (2) with node 18.

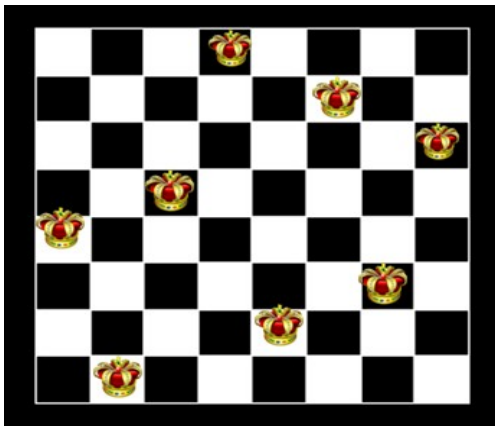
We observe that the path to answer node is (2 4 1 3)

6.4 8-Queens Problem

Similar to 4Queens problem, in 8Queens problem also has the same objective that no two queens should place in the same row, same column or diagonal position.



a) Solution is (4, 7, 3, 8, 2, 5, 1, 6)



b) Solution is (4, 6, 8, 3, 1, 7, 5, 2)

N-Queens problem

In implementing the n – queens problem we imagine the chessboard as a two-dimensional array A (1: n, 1: n). The condition to test whether two queens, at positions (i, j) and (k, l) are on the same row or column is simply to check I = k or j = l. The conditions to test whether two queens are on the same diagonal or not are to be found.

Observe that

i) For the elements in the upper left to lower right diagonal, the row -column values are same or row- column = 0,

E.g. 1-1=2-2=3-3=4-4=0

ii) For the elements in the upper right to the lower left diagonal, row + column value is the same e.g.

1+4=2+3=3+2=4+1=5

Thus two queens are placed at positions (i, j) and (k, l), then they are on the same diagonal only if

i – j = k - l or i + j = k+ l

(or)

j - l = i - k or j - l = k – I

Two queens lie on the same diagonal if and only if |j – l| = |i - k|

(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 1)	(3, 2)	(3, 3)	(3, 4)
(4, 1)	(4, 2)	(4, 3)	(4, 4)

Time complexity: O (n!)

6.5 Sum of subsets problem

If there are n positive numbers given in a set. Then the desire is to find all possible subsets of the contents of which to add onto a predefined value M. In other words, let there be n elements given by the set W= (W1, W2… W3) then find out all the subsets from whose sum is M.
Briefly its goal is to maximize the total value of the solution (M) items while not making the total weight exceed W. If we sort the weights in non decreasing order before doing the search, there is an obvious sign telling us that a node is non promising.

Let *total* be the total weight of the remaining weights, a node at the i^{th} level is non promising if ***Weight* + *total* > W**

Visualize a tree in which the children of the root indicate whether or not value has been picked (left is picked, right is not picked).

Sort the values in non-decreasing order so the lightest value left is next on list.

Weight is the sum of the weights that have been included at level i

Let *weight* be the sum of the weights that have been included up to a node at level i. Then, a node at the i th level is non promising if ***weight* + wi +1 > W**

Simple choice for the bounding Function is Bk (X1 … Xk) = true iff

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

Clearly x1 …xk can not lead to an answer node if this condition is not satisfied.

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} > m$$

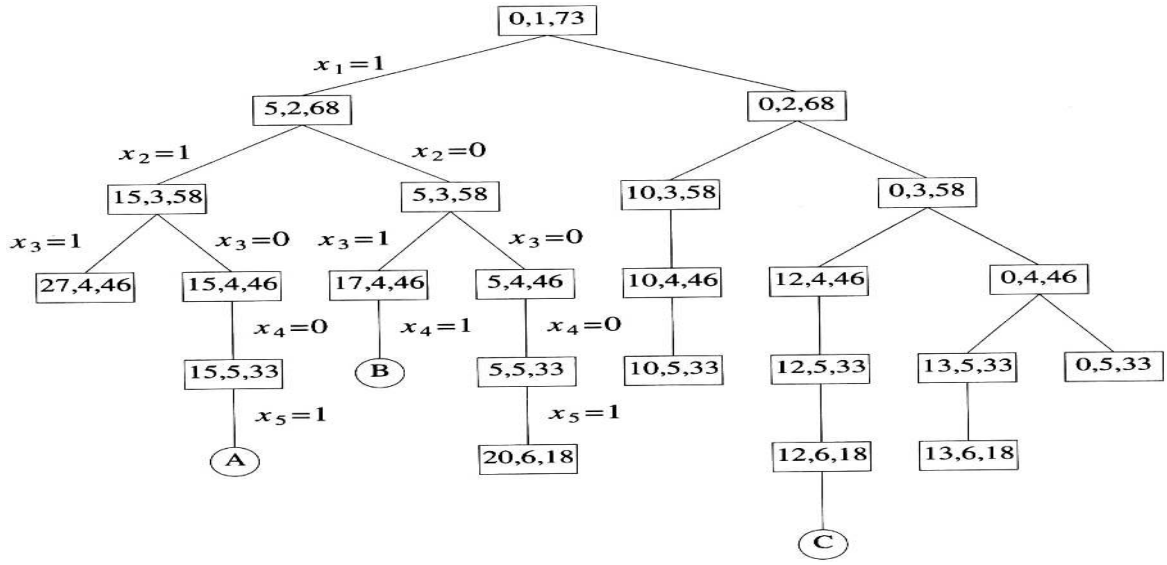
Assuming wi's in non decreasing order, (x1... xk) cannot lead to an answer node if

So, the bounding functions we use are therefore

$$B_k(x_1,\ldots,x_k) = true \text{ iff } \sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

$$and \quad \sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m$$

Example:
n=6, w [1:6] = {5, 10, 12, 13, 15, 18}, m=30



6.6 Graph coloring:

Let G be a graph and m be a positive integer.

The problem is to color the vertices of G using only m colors in such a way that no two adjacent nodes / vertices have the same color.

It is necessary to find the smallest integer m and m is referred to as the chromatic number of G. A special case of graph coloring problem is the four color problem for planar graphs.

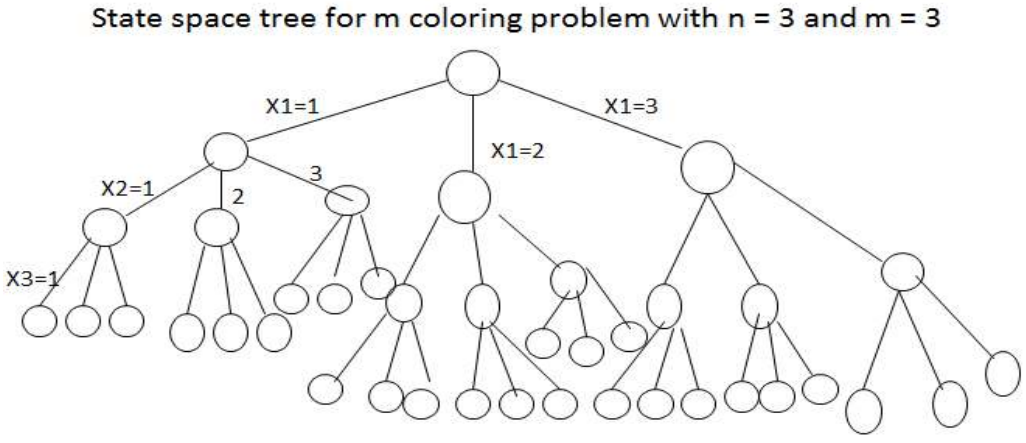
A graph is planar iff it can be drawn in a plane in such a way that no two edges cross each other.

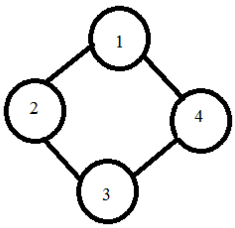
4- colour problem for planar graphs. Given any map, can the regions be colored in such a way that no two adjacent regions have the same colour with only four colors?

A map can be transformed into a graph by representing each region of map into a node and if two regions are adjacent, then the corresponding nodes are joined by an edge.

For many years it was known that 5 colors are required to color any map.

After a several hundred years, mathematicians with the help of a computer showed that 4 colors are sufficient.



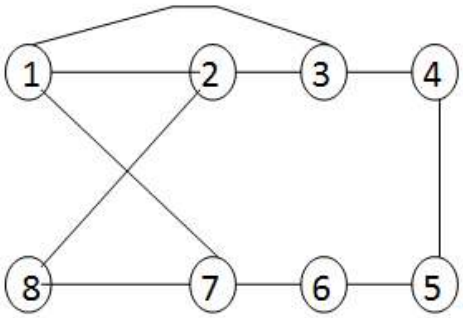


Example:

- Program and run m coloring algorithm using as data the complete graphs of size n=2, 3, 4, 5, 6 and 7. Let the desired number of colors be k=n and k=n/2

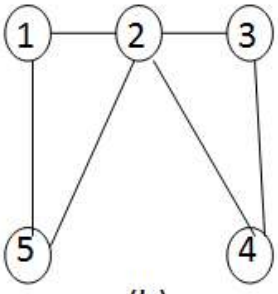
6.6 Hamiltonian cycles

- Let G = (V, E) be a connected graph with n vertices.
- A Hamiltonian cycle is a round path along n edges of G which visits every vertex once and returns to its starting position.
- The tour of a traveling salesperson problem is a Hamiltonian cycle.
- A tour may exist or not.



(a)

Hamiltonian Cycle is 1,2,8,7,6,5,4,3,1



(b)

No Hamiltonian Cycle

The backtracking solution is a vector ($x_1 \ldots x_n$) where x_i represents the i^{th} visited vertex of the cycle.

To avoid printing of the same cycle n times we require X(1) = 1 (as 128765431, 287654312, 87654312)

We compute X (k) given (x_1…..x_{k-1}) have already been chosen.

Two procedures NEXTVALUE(k) and HAMILTONIAN are used, to find the tour.

We initialize Graph (1:n, 1:n) and X(2:n)←0, X(1)←1 and start with HAMILTONIAN (2).

State space tree

Put the starting vertex at level 0 in the tree; call it the zero th vertex on the path.
At level 1, consider each vertex other than the starting vertex as the first vertex after the starting one.
At level 2, consider each of these same vertices as the second vertex, and so on.
Finally, at level n-1, consider each of these same vertices as the (n-1) st vertex

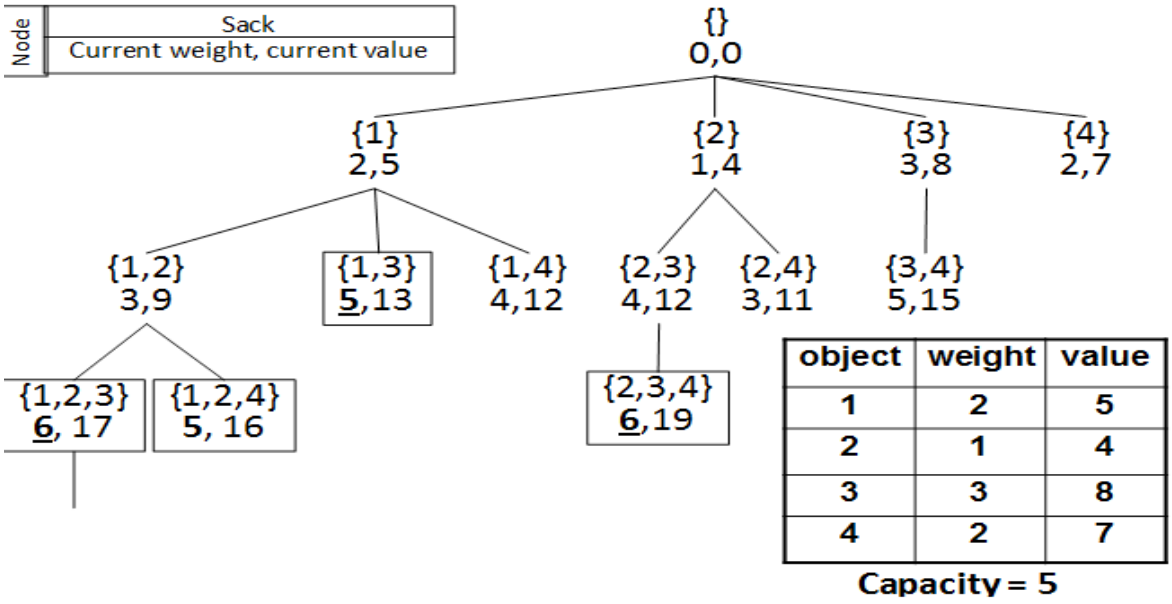
1. The *i* th vertex on the path must be adjacent to the (*i*–1) st vertex on the path.
2. The (*n*–1) st vertex must be adjacent to the 0th vertex (the starting one).
3. The *i* th vertex cannot be one of the first (*i*–1) vertices.

Example
Let n = 8
X (1) = 1, HAMILTONIAN(2) i.e. H (2) is called, so NEXTVALUE(2) i.e. N(2) is called.
Initially X (2) = 0

X (2) = 0+1 mod 9 = 1 but X (1) = X(2) so loop is repeated and X(2) = 2 mod 9 = 2
X (1) ≠ X (2) and j=k=2, k < 8 so return 2
NV(3) = 8 as Graph(2,3), Graph(2,5) Graph(2,6),Graph(2,7),Graph(2,4) are false.

Thus NV(4) = 7,NV(5) = 6,NV(6) = 5 NV(7) = 4, NV(8) = 3.
At NV (8), k = 8 and GRAPH(X (8), 1) is satisfied. Thus the cycle is printed.

6.7 Knapsack problem using Backtracking:



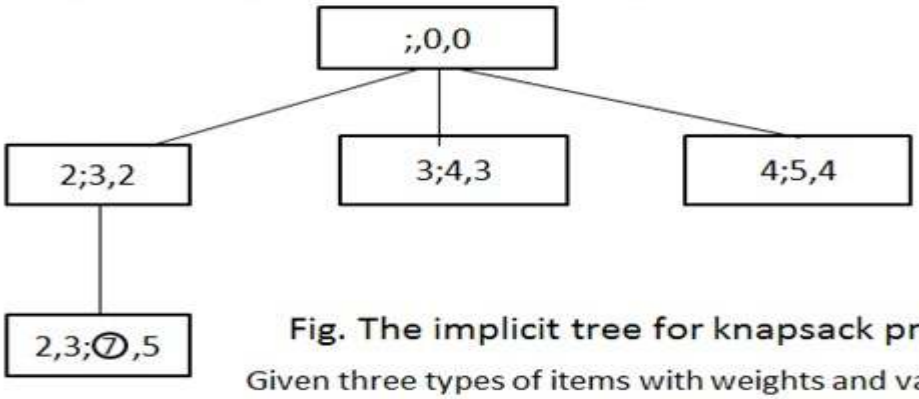


Fig. The implicit tree for knapsack problem

Given three types of items with weights and values and

Given three types of items with weights and values and knapsack capacity w=5.In the above we backtrack one step and find that new addition (2, 4; 8, 6) will also violate the knapsack capacity. In each node left hand side of semicolon is weight chosen, right hand side of semicolon total value and next total weight which is taken in increasing order

T	W	v
T1	2	3
T2	3	4
T3	3	4
T4	4	5

Exercise

1. Given three types of items with the weights and values are

   T = <T1, T2, T3>
   Wi = <1, 4, 5>
   Vi = <4, 5, 6>

****