UNIT-6

Backtracking

Concept [Imp.]

Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem. It is general algorithm for finding solution to some computational problem. We have set of several choices. If one choice from set of choices proves incorrect, computation backtracs at the point of choice and tries another choice. In backtracking we use recursion in order to explore all the possibilities until we get the best result for the problem. Backtracking is a depth-first search with any bounding function.

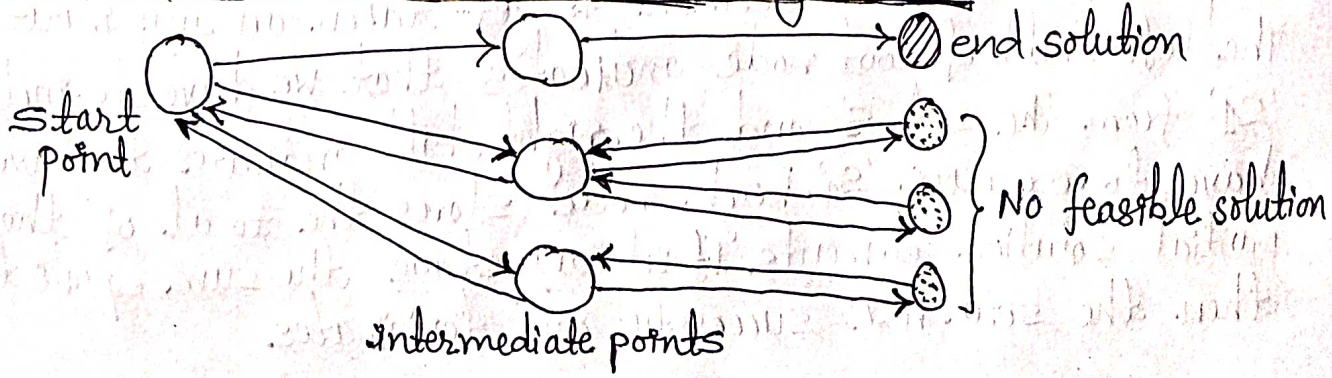
Advantages:

→ It is a step-by-step representation of a solution to a given problem, which is very easy to understand.
→ It has got a definite procedure.
→ It is independent of programming language.
→ Code size is usually small.
→ It is easy to debug as every step has its own logical sequence.

Disadvantages:

→ It is time consuming computer program.
→ Multiple function calls are expensive.
→ Inefficient when there is lot of branching from one state.
→ It requires large amount of space as each function state needs to be stored on system stack.
→ Backtracking is hard to simulate.

Example to demonstrate backtracking: [Imp.]



Recursion vs. Backtracking [Imp]

Recursion	Backtracking
i) Recursion is a method of solving problems that involves breaking a problem down into smaller and smaller sub-problems until can be solved trivially.	i) Backtracking is a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.
ii) It involves a function calling itself.	ii) It involves algorithm for finding solution to ~~computational~~ computational problem.
iii) Recursion is like bottom-up process.	iii) Backtracking is like a top-down process.
iv) For example: recursion is used in recursive functions such as factorial, gcd etc.	iv) For example: backtracking is used in algorithms such as BFS, DFS etc.

#Backtracking Algorithms:

1) Subset-sum Problem:

The Subset-sum problem is to find a subsets of the given set $S = \{S_1, S_2, \ldots, S_n\}$. We assume that the elements of the given set are arranged in increasing order.

$$S_1 \leq S_2 \leq S_3 \ldots \leq S_n.$$

In this approach we have binary tree as implicit tree in which root of tree is selected in such a way that represents that no decision is yet taken on any input. The left child of root node indicates that we have to include 'S1' from the set 'S' and the right child indicates that we have to execute 'S1'. Each node stores the total of the partial solution elements. If at any stage the sum equals to 'X' then the search is successful and terminates.

The dead end in the tree appears only when either of the two inequalities exists:

i) The sum of s' is too large
   i.e, $s' + s_j + 1 > X$

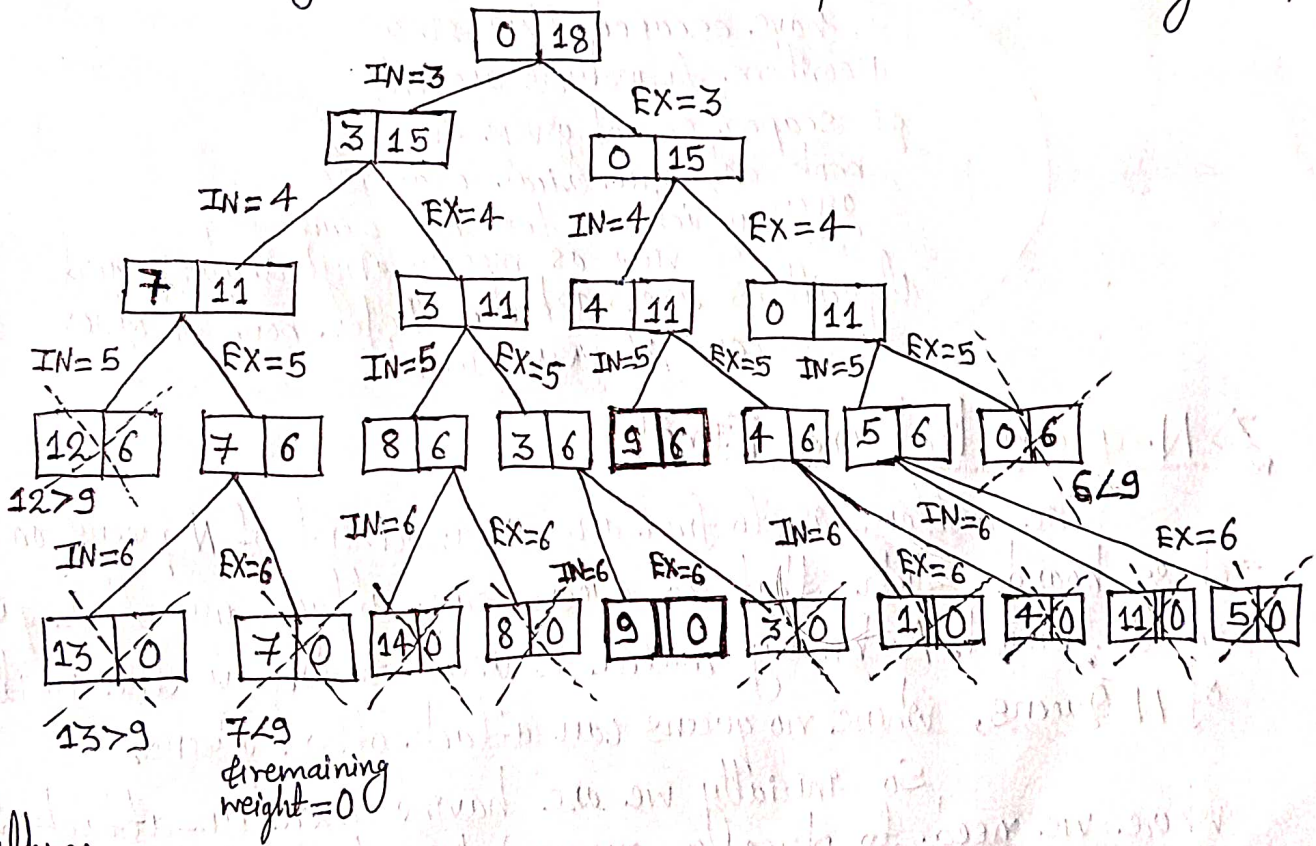
ii) The sum of s' is too small
   i.e, $s' + \sum_{j=i+1}^{n} s_j < X$

Example: Given a set $S = \{3, 4, 5, 6\}$ and $X = 9$. Obtain the subset sum using backtracking approach.

Solution: Initially $S = (3, 4, 5, 6)$ and $X = 9$.

$S' = \{\phi\}$

The implicit binary tree for the subset sum problem is shown in fig below;



Algorithm:

1. Start
2. If index == array.length then
   Return false
3. If array[index] == sum then
   Return true

4. Iterate given array from index to array.length
   If array[i] > sum then,
      Don't do anything take next element from array.
   If array[i] == sum then
      Return true.
   Recursively call with index +1 and sum - array[i].
   If last recursive call was success then
      Return true.
5. Return false.
6. Stop.

2) Zero-One Knapsack problem with backtracking approach:

I have escaped this since algorithm & analysis are escaped or not given in book only numerical example given which is less imp exam point of view as mostly ~~about~~ analysis and algorithms are asked. Refer book or other sources if you want.

3) N-queen Problem:

The problem is to find an arrangement of N queens on a chess board, such that no queen can attack any other queens on the board. A binary matrix is used to display the positions of N Queens, where no queens can attack other queens.

So initially we are having n×n un-attacked cells where we need to place n queens. Lets place the first queen at a cell (i,j), so now the number of un-attacked cells is reduced, and number of queens to be placed is n-1. Place the next queen at some un-attacked cell. This again reduces the number of un-attacked cells and number of queens to be placed becomes n-2. Continue doing this as long as following conditions hold.

→ The number of un-attacked cells is not 0.
→ The number of queens to be placed is not 0.

If the number of queens to be placed becomes 0, then it's over, we found a solution. But if the number of un-attacked cells become 0, then we need to backtrack, i.e, remove the last placed queen from its current cell, and place it at some other cell. We do this recursively.

Algorithm:

1. Start
2. Place the queens column wise, start from the left most column.
3. If all queens are placed.
   Return true and print the solution.
   Else
   a) Try all the rows in the current column.
   b) Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
   c) If placing the queen in above step does not lead to solution, BACKTRACK, mark the current cell in solution matrix as 0 and return false.
   d) If placing the queen in above step leads to solution return true.
4. If all the rows are tried and nothing worked, return false and print NO SOLUTION.
5. Stop.

Analysis: Solution of N Queen problem using backtracking checks for all possible arrangements of N Queens on the chessboard. And then checks for validity of the solution. Now number of possible arrangements of N Queens on N×N chessboard is N!. So average, best & worst case complexity of the solution is O(N!).