Unit-6 : Backtracking

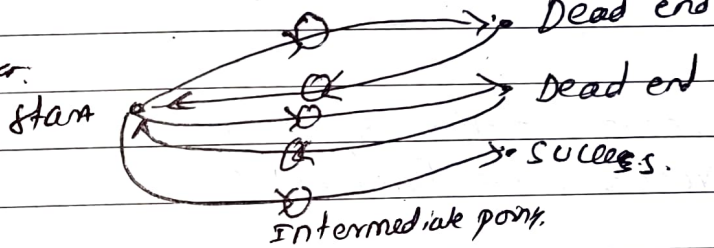
DATE

6.1 Introduction

Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem. We have a set of choices. If one of choice proves incorrect, computation backtracks at the point of choice and tries another choices until it finds the right choice.

Here, we use the recursion in order to explore all the possibilities until we get the best result for the problem. It is a depth first search with any bounding function. We can say that the backtracking is used to find all possible combination to solve an optimization problem.

Pictorial representation:



Advantages:

→ Represents a step-by-step solution of a problem, so easy to understand.
→ It can find all possible solutions of a problem.
→ It has got a definite procedure.
→ It is independent of programming language.
→ It is easy to debug as every step has got its own logical sequence.

Disadvantages:

→ Can be time consuming & may lead to a large number of branches to explore, especially in problems with a large no. of possible solutions.
→ It doesn't come with heuristics to guide the search.
→ It is hard to simulate by human simulate.
→ Requires a large amount of space as each function needs to be stored on system stack.

classmate

PAGE

DATE

* Recursion vs Backtracking

Recursion	Backtracking
i) Recursion involves breaking down a problem into subproblems and solving the subproblems to get the solution of the problem.	i) It involves searching every possible combination in order to solve a computational problem. It uses the concept of recursion.
ii) Recursion can be efficient relatively easy to understand and implement.	ii) It is also relatively easy to understand but it can be slow for large inputs & difficult to terminate the search early if a solution is found.
iii) Some problems that uses recursion are Factorial, Gcd	iii) Backtracking is used in algorithms such as ~~BFS~~, DFS, Sodoku or puzzle.
iv) It is like a bottom up process.	iv) It is like a top down process
v) In recursion, function calls itself until it reaches a base case	v) In backtracking, you explore all the possibilities until you get the best result for the problem. (Route)

classmate

PAGE

DATE

6.2. Backtracking Algorithms.

* Subset Sum Problem

Given a set of integers $S = \{s_1, s_2, s_3, \ldots s_n\}$ (arranged in ascending order) and a target integer 't', one must determine the subset of the integers in the set that add up to the target integer. This problem is known as Subset sum problem

In backtracking approach we use a binary tree as implicit tree in which root of the tree is selected such that the sum is 0 ie. no decision is yet taken on any input. The left child of root node indicates we include s_i from the set ie ($s_i = 1$) & the right child indicates we exclude s_i (ie $s_i = 0$). Each node stores the sum of the partial solution elements. If at any stage, the sum equals to target 't' or $> t$, the search is terminated. We also terminate the search, when the sum is so small that including rest of the elements will still result in sum less than the target

ie. dead end is when..

Sum s' is too large ie. $s' + s_{i+1} > t$

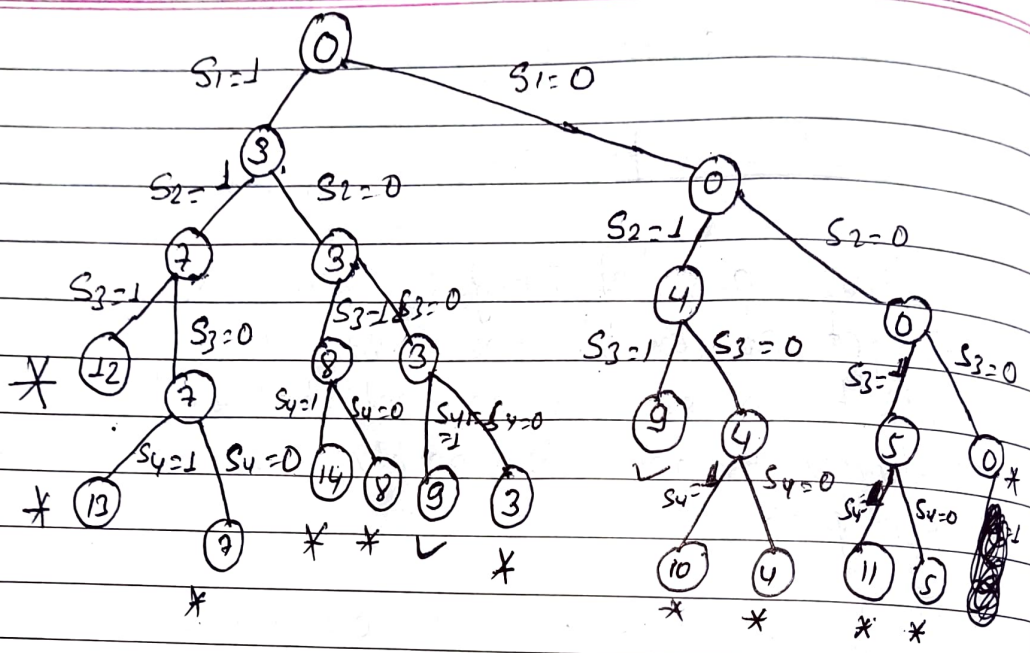
Sum s' is too small ie. $s' + \sum_{j=i+1}^{n} s_j < t$

Ex: Given a set $S = \{3, 4, 5, 6\}$ and $X = 9$ (target). obtain the subset sum using backtracking approach

a) Initially $S = (3, 4, 5, 6)$ & $X = 9$

$S' = \{\phi\}$.

The implicit binary tree for the subset sum is:



Algorithm:

array length

```
subset_sum(set, n, sum)
{ n = len(set)
  if (sum==0) return true
  if(n==0 and sum!=0) return false.
if (set[n-1] > sum) return subset_sum(set, n-1, sum)
    return subset_sum(set, n-1, sum)
or subset_sum (set, n-1, sum-set[n-1])

if (subset_sum(set, n, sum) == true)
  print( Found a subset with given sum
else print( No subset with given sum)
}
```

Analysis: For each element, function has two choice either include or exclude it. This leads to 2^n possible subsets that need to be checked & for each subset function takes $O(1)$ time

$\therefore T(n) = O(2^n) \times O(1)$

$= O(2^n)$

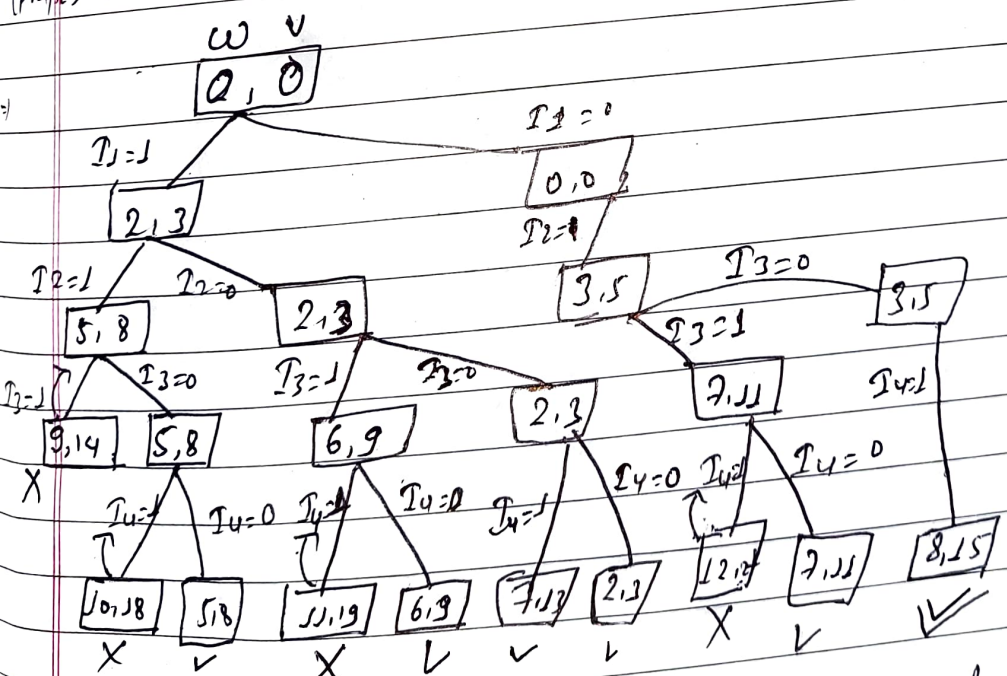
DATE

Zero-one Knapsack problem with backtracking approach

Given a set of items, each with a weight and a value(profit), determine the number of each items to include in a collections so that the total weight is less than or equal to a given limit & total value (profit) is as large as possible. i.e. maximize the total profit earned.
This problem is known as 0/1 knapsack problem

Ex: Find the maximum profit earned by using 0/1 knapsack problem of Backtracing for the following data items with their respective weight & value.

Items = 1 2 3 4
weight = 2 3 4 5 & Knapsack of weight (W) = 8
(profit) values = 3 5 6 10



Hence, maximum profit earned = 15 with full weight

PAGE

classmate

DATE

N-Queen Problem

The N-Queen problem is a classical problem, where the goal is to place N queens on a NxN chess board in such a way that no queens threaten each other. Queen can move rowwise, columnwise & diagonally

One common approach of solving this problem is by using backtracking. It can check for conflicts between queens by checking if any two queens share the same row, column or diagonal. The backtracking algorithm explore all possible solutions & once a solution is found, the algorithm can stop & return the solution.

Algorithm:

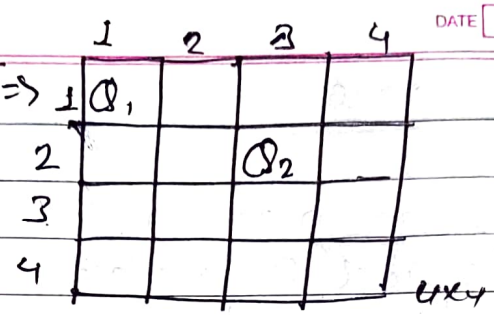
1. Start
2. Place the queens row wise, start from the top most row
3. If all queens are placed
   a) return true and print the solution matrix
4. Else a) Try all the columns (cells) in the current row
   b) Check if queen can be placed here safely, if yes, mark the current cell in solution matrix as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

PAGE

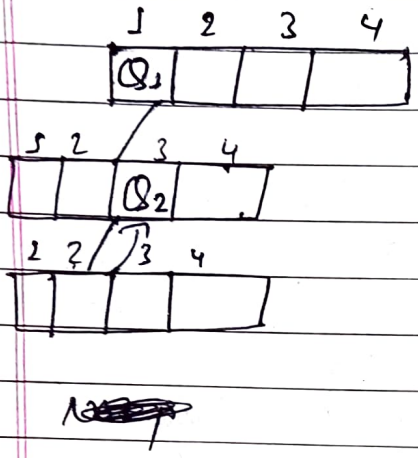
classmate

DATE

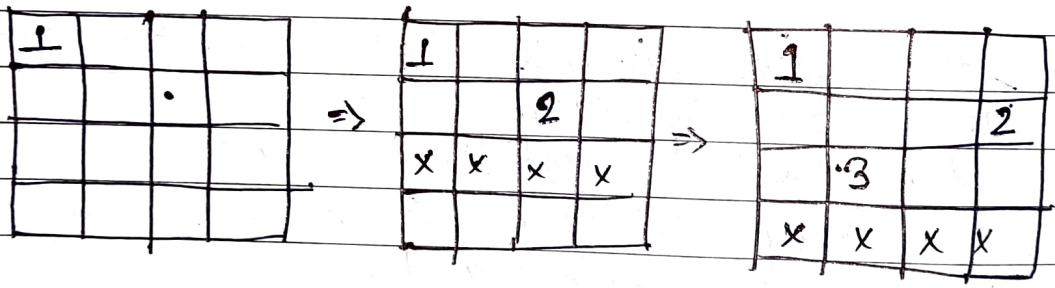
Ex: Let's take N=4 ⇒



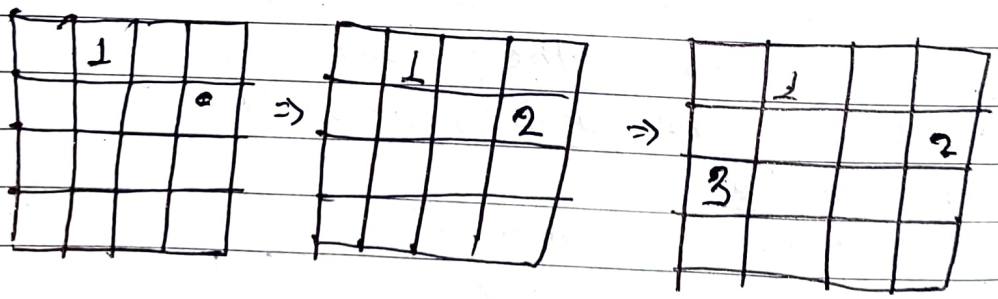
4×4

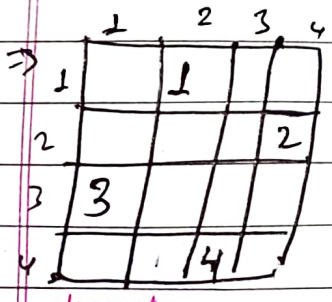


Ex: Let's take N=4:



⇒





which is one of the solution of 4-Queen problem of sequence - A

(2, 4, 1, 3)

1st row 2nd row 3rd row 4th row

Or in pairs (1,2), (2,4), (3,1), (4,3)

PAGE

DATE

Analysis:
No. of possible arrangements of N queens in NxN chess board = N!

∴ The best, average & worst case is T(N) = O(N!)

Here, the best case occurs if we find our solution before exploiting all possible arrangements. This depends on our implementation. And if we need all the possible solution, then T(N) = O(N!) for all case.