DAA

Backtracking Algorithm.

Introduction:

Backtracking is an algorithmic technique where the goal is to get all solutions to a problem using the brute force approach. It consists of building a set of all the solutions incrementally. Since a problem would have constraints, the solutions that fail to satisfy them will be removed.

It uses recursive calling to find a solution set by building a solution step by step, increasing levels with time. In order to find these solutions, a search tree named state-space tree is used. In a state-space tree, each branch is a variable, and each level represents a solution.

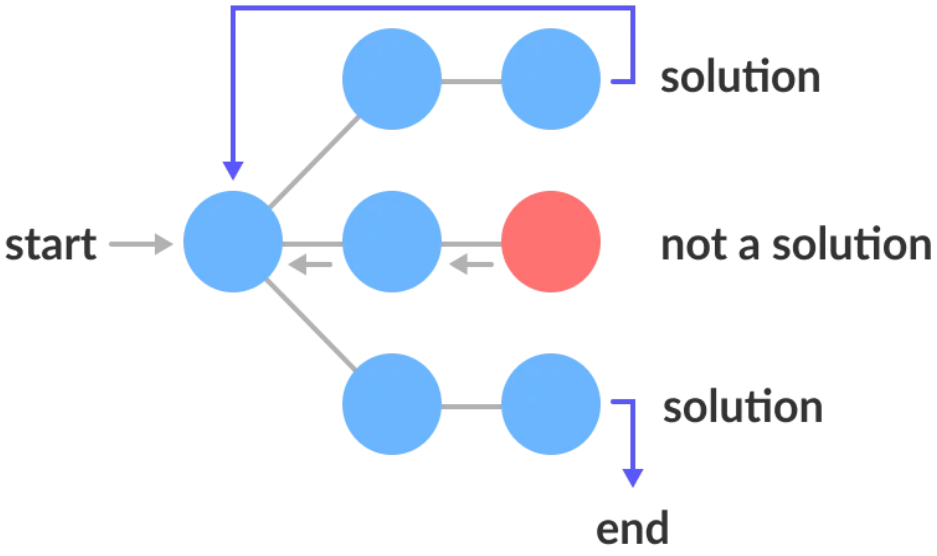
A backtracking algorithm uses the depth-first search method. When it starts exploring the solutions, a bounding function is applied so that the algorithm can check if the so-far built solution satisfies the constraints. If it does, it continues searching. If it doesn't, the branch would be eliminated, and the algorithm goes back to the level before.

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using recursive algorithms, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.

State Space Tree

A space state tree is a tree representing all the possible states (solution or nonsolution) of the problem from the root as an initial state to the leaf as a terminal state.



When to Use a Backtracking Algorithm

The backtracking algorithm is applied to some specific types of problems. For instance, we can use it to find a feasible solution to a decision problem. It was also found to be very effective for optimization problems.

For some cases, a backtracking algorithm is used for the enumeration problem in order to find the set of all feasible solutions for the problem.

On the other hand, backtracking is not considered an optimized technique to solve a problem. It finds its application when the solution needed for a problem is not time-bounded.

There are three types of problems in backtracking –

Decision Problem – In this, we search for a feasible solution.

Optimization Problem – In this, we search for the best solution.

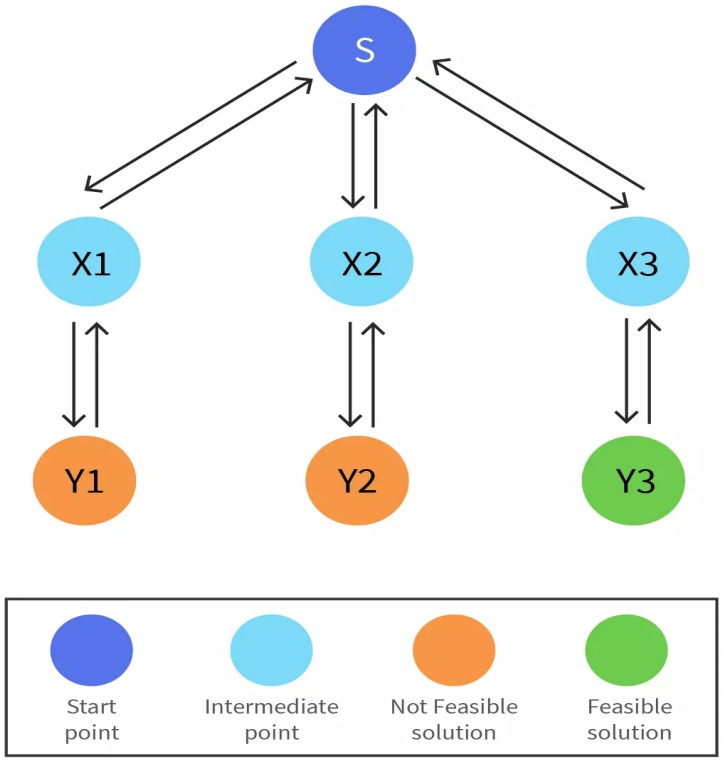
Enumeration Problem – In this, we find all feasible solutions.

Algorithm:

```
Backtrack(x)
    if x is not a solution
        return false
    if x is a new solution
        add to list of solutions
    backtrack(expand x)
```

How Backtracking Algorithms Works?

For Instance: Consider the following space state tree. With the help of the below space state tree let's understand how the backtracking algorithm actually works.



We start from S which is the starting point of our problem. We go to find a solution to Y1 from intermediate point X1 and then we find that Y1 is not a feasible solution to our problem therefore we backtrack and go back to X1 via Y1 then go back to S and then again try to find out the feasible solution by following another path S->X2->Y2 and again we will find that Y2 is not a feasible solution so we will again return to S by following path Y2->X2->S and then we will ultimately reach out to feasible solution Y3 by following the path S->X3->Y3.

Use cases

As was already mentioned in the introduction, this algorithm can be applied in different niches in the computer science area. So let's talk about the use cases for the Backtracking algorithm:

- Cybersecurity

If you are a cybersecurity engineer, a student, or even a hacker, Backtracking might be a useful tool to solve some problems. You can use it to perform searches in a file system, you can also design algorithms to find breaches, and the most common use case of this algorithm for cybersecurity is to brute force passwords.

- Compilers

To build a compiler you need to study a lot, plan the steps and define which strategies you will use. One mandatory step of a compiler project is the planning of the parser (syntactic analyser). At this stage of planning, you must define which parser implementation you will use, one of them is top down parser with backtracking. This implementation is not very popular because it has a very-high computational cost, but depending on your needs, you can choose it.

- Artificial intelligence

In artificial intelligence, the Backtracking algorithm is used in decision, optimization and enumeration situations. Backtracking is a common solution, but AI researchers around the world are trying to build models and artificial intelligences without using Backtracking. They are looking for a more lightweight solution, with a lower computational cost.

Pros and cons

Pros

- Backtracking can almost solve any problems, due to its brute-force nature.
- Can be used to find all the existing solutions if there exists for any problem.
- It is a step-by-step representation of a solution to a given problem, which is very easy to understand.
- Very easy to write the code, and also to debug.

Cons

- It is very slow compared to other solutions.
- Depending on the data that you have, there is the possibility to perform a very large search with Backtracking and at the end don't find any match to your search params.
- Very-high computational cost, Backtracking is a recursive algorithm that consumes a lot from the memory and from the processor.
- If you need an optimized algorithm, that is able to handle a large volume of data and find all the possible solutions consuming less computational resources and in a shorter interval of time, you may consider using the branch-and-bound algorithm.

Application of Backtracking:

Creating word game:

Problem Statement :

Given a matrix of characters with n rows and m columns, and few words. We have to find out if the words can be formed by a series of adjacent elements. Print "Yes" if words can be formed with adjacent elements else print "No".

Solution Approach :

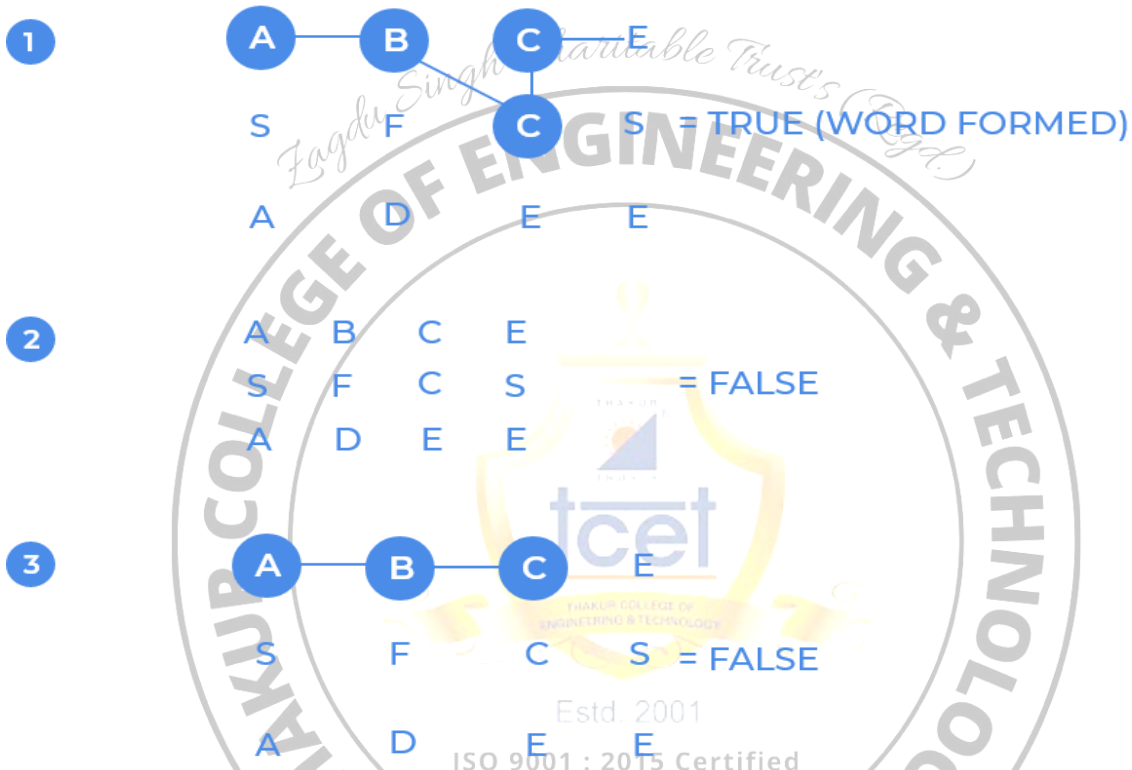
Introduction :

Idea is to iterate for every character in the matrix and check if there is any adjacent character which matches with the character of the word, if it does keep checking for the next character of the word until the complete word is formed with adjacent characters in the matrix.

Array =

$$\begin{bmatrix} A & B & C & E \\ S & F & C & S \\ A & D & E & E \end{bmatrix}$$

words = "ABCCDE", "SEE", "ABCB"



Description:

We will use backtracking to solve the above problem.

Backtracking is the approach to solve the problem by testing all possible answers (by breaking the problems into subproblems). If any subproblem does not fit the given constraint then we discard the complete subproblem (backtrack), move a step back then try other remaining possible combinations. Backtracking algorithm is generally exponential in time.

As discussed above we will iterate through each character of the matrix and see if any of the characters matches with the character of the word. If it matches then we will recursively check for the next character in the word if it matches. We will

return true if all the characters in the words match, else return false. We also have to keep track of the checked characters so that any character must not be checked twice. If some of the subproblems (recursive calls) fails, we will mark the characters as unvisited again so we can again use them for checking.

Algorithm : bk():

if index==wordsize, return TRUE.

if i<0 or j=n or j>=m, return FALSE.

store the current character, temp = a[i][j].

change the current character to mark it visited or used.

recursively call bk() for all four adjacent cells, if any of the calls return true, assign the original character to a (a[i][j]= temp) & return TRUE..

if none of the previous calls has returned true, return FALSE

Conclusion:

Backtracking is a technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time.The backtracking technique is generally used in cases where there are possibilities of multiple solutions.It is better than the brute force approach that tries out all the possible solutions and chooses the best possible ones out of them.

Group members: 57 Rishabh Tiwari

58 Yoshit Verma

59 Vedanti Wadekar

60 Nidhi Worah