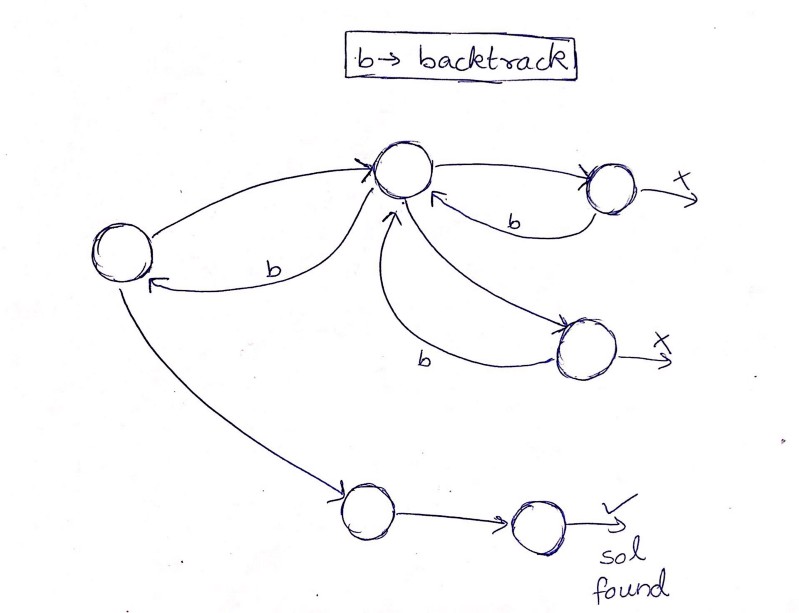
<https://medium.com/leetcode-patterns/leetcode-pattern-3-backtracking-5d9e5a03dc26>

A very important tool to have in our arsenal is backtracking, it is all about knowing when to stop and step back to explore other possible solutions. In today’s post we’ll explore the common pattern in solving backtracking problems and set up the stage to dive into dynamic programming (DP) problems next. It is amusing how a small change in the problem can change the solution from DP to backtracking and understanding this will help us save time.

Backtracking can be seen as an optimized way to brute force. Brute force approaches evaluate every possibility. In backtracking you stop evaluating a possibility as soon it breaks some constraint provided in the problem, take a step back and keep trying other possible cases, see if those lead to a valid solution.



So in fact, it’s kinda like a depth-first search(DFS) with an added constraint that we stop exploring the subtree as soon as we know for sure that it won’t lead to valid solution. The problems that can be solved using this tool generally satisfy the following criteria :

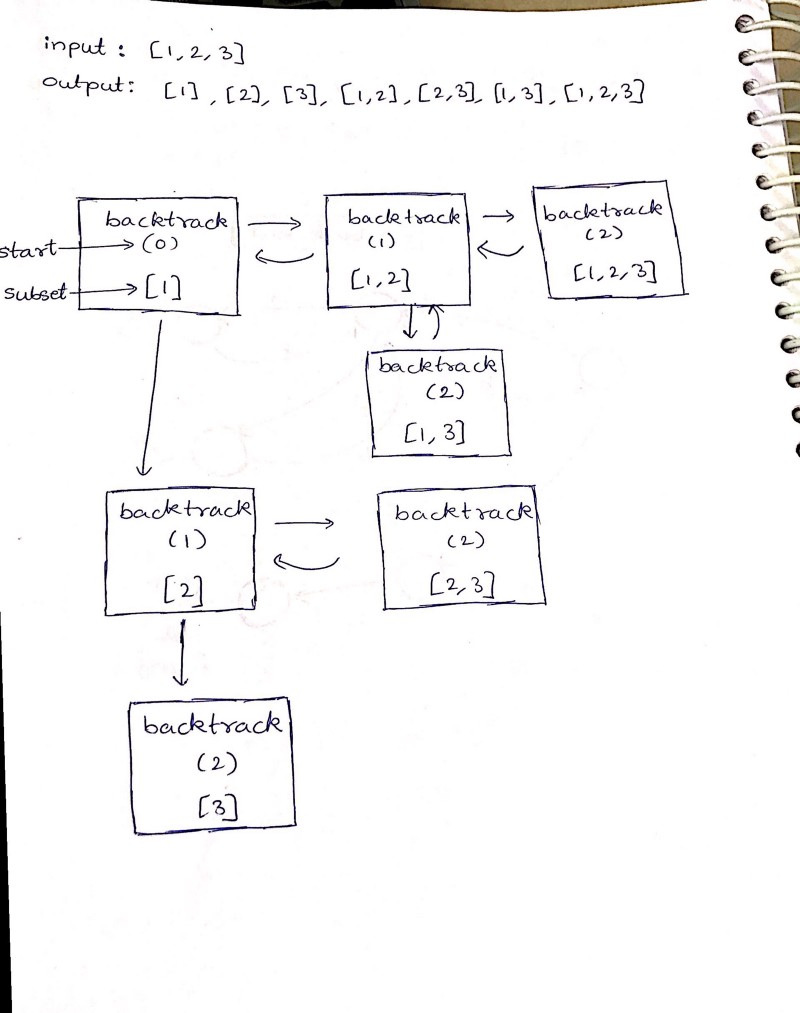
1. You are explicitly asked to return a collection of all answers.
2. You are concerned with what the actual solutions are rather than say the most optimum value of some parameter. (if it were the latter it’s most likely DP or greedy).

[**78. Subsets**](https://leetcode.com/problems/subsets/description/)

We’ll use this problem to get familiar with the recursive backtracking pattern.

The problem is to find the powerset of a given set, so we simply need to collect all possible subsets of a set. This could be done in a variety of ways and backtracking is one way to go, also since there are no constraints provided we simply explore all cases (all subsets).

A subset can either have an element or leave it out giving rise to 2^n subsets.



Take a moment to absorb the magic happening in the loop and that’s everything you’ll ever need to solve a backtracking problem. It was confusing to me at first but it’s an amazing pattern. Drawing the flow of the recursive function helped me wrap my head around what is going on.

Also here is my alternative solution to the same problem which uses the partially formed output to generate the full output incrementally. It’s basically deriving the complete solution from solutions of smaller problems, does it ring a bell?

#### [****90. Subsets II****](https://leetcode.com/problems/subsets-ii/description/)

Same problem with the added constraint that the set may contain duplicates but the output power set should not contain duplicate subsets.

Approach: Sort the given array beforehand and skip over duplicates while backtracking, essentially a simple 2 line change in the previous solution.

#### [****39. Combination Sum****](https://leetcode.com/problems/combination-sum/description/)

Notice that we’ll have to explore many cases and there is no “smart” way to avoid that, the only smart thing we could do is to stop exploring a case as soon as we know it won’t lead to the solution and so this is a backtracking problem.

The solution is entirely same as subsets solution, only with a slight modification that we have a constraint included: the sum of the final collected combination should equal target.

#### [40. Combination Sum II](https://leetcode.com/problems/combination-sum-ii/description/)

This problem bears the same relation to the previous problem as subsets-2 had with subsets, as such it should be no surprise that the same strategy works!

Honestly, visualizing the flow of the recursive function above is kinda trippy. Once you get comfortable writing this back and forth flow, backtracking problems are really easy. Also as I was writing this article I had an idea to make it simpler, so here is a simpler version of the subsets solution.(could be extended for other solutions in this post as well)

Time for one final problem without which our discussion on backtracking would be considered incomplete, one of the classic computer science problems which gave birth to this paradigm.

#### [51. N-Queens](https://leetcode.com/problems/n-queens/description/)

We try placing queens column by column. Place a queen, go to next column and try placing another queen such that it doesn’t face a queen in the same row or diagonals ( which is checked in validateSpot method ), and keep going. If we encounter an invalid spot we backtrack and keep trying other spots in that column vertically.

The validateSpot method can be made more efficient by using arrays to store the diagonals and rows already occupied. If you are interested, do check out this [solution](https://leetcode.com/problems/n-queens/discuss/19808/Accepted-4ms-c++-solution-use-backtracking-and-bitmask-easy-understand.)

Finally the point I mentioned earlier, when does a backtracking problem convert to a DP one? Let’s take an example:

* Return all ways to climb stairs, jumps allowed in steps 1 -> k
* Count ways to climb stairs, jumps allowed in steps 1-> k

The first would require backtracking as we actually need all the ways, while the 2nd one could be done using DP optimally and is similar to how we optimize calculating Fibonacci numbers with memoization.

In the next post we’ll see solutions to these problems as well as explore other such cases (the standard rod cutting problem vs the subsets problem above). Understanding when to use DP is in itself a major issue. At this point I would like to point out the strong bond between recursion, backtracking, depth first search, and dynamic programming. (mega pattern if you will! )

The next few posts will be on solely focused on decoding DP patterns as many of you have requested the same. Stay tuned for upcoming posts!

References:

* [discuss thread on backtracking](https://leetcode.com/problems/subsets/discuss/27281/a-general-approach-to-backtracking-questions-in-java-subsets-permutations-combination-sum-palindrome-partitioning)
* [backtracking vs dfs](https://stackoverflow.com/questions/1294720/whats-the-difference-between-backtracking-and-depth-first-search)
* [geeksforgeeks backtracking](https://www.geeksforgeeks.org/backtracking-algorithms/)