App Academy

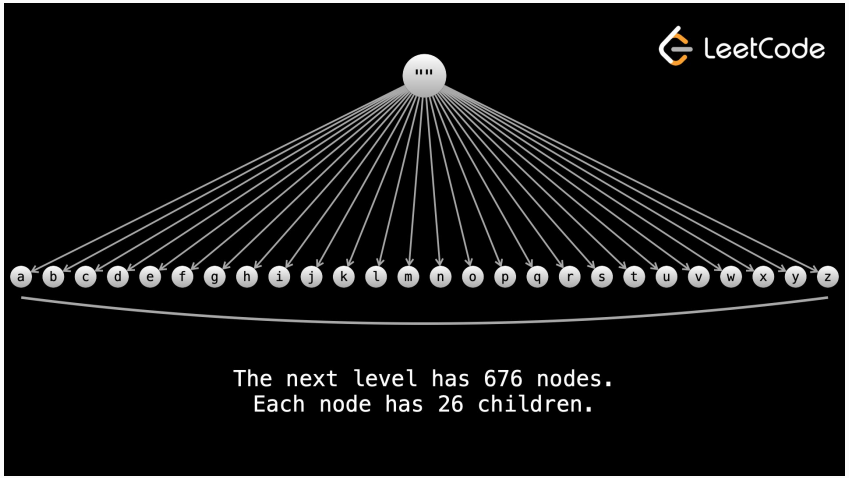
Backtracking

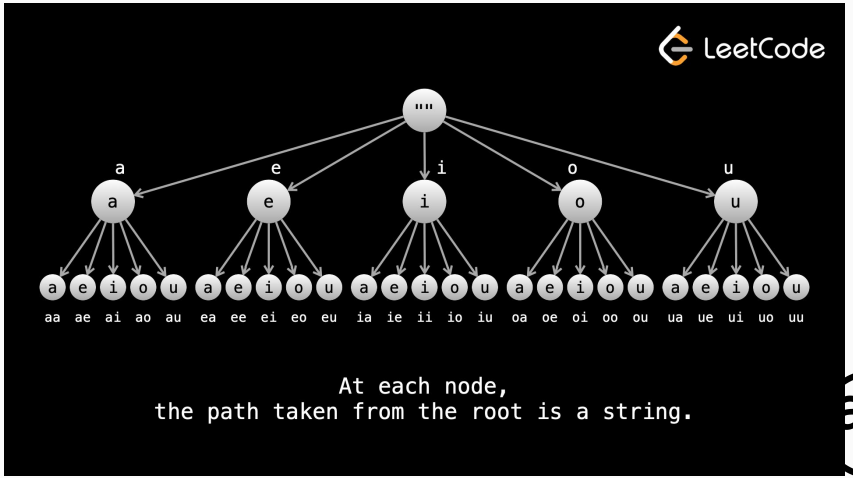
Backtracking

- Backtracking is a algorithmic technique where we solve problems recursively by “building candidates” for solutions and then abandon those candidates (“backtrack”) once we determine that candidate can no longer yield a valid solution.
- Built on DFS and an optimization of “exhaustive search”, where we search through all possible candidates.
- Backtracking is an optimization that involves abandoning a "path" once it is determined that the path cannot lead to a solution (in other words, our base case)
- This is an important concept because some problems can only be (reasonably) solved via backtracking.
- Most common type of problem that can be solved with backtracking is "find all possible ways to do something”
- Backtracking problems run in exponential runtime. example: $O(2^n)$

Example

Let's take a simple example where we're building all possible strings with a length of n. To check all possible strings, we'd be running a $O(26^n)$ runtime. But what if vowels-only was a constraint? We can discard all non-vowel subtrees and improve runtime to $O(5^n)$.





Backtracking skeleton

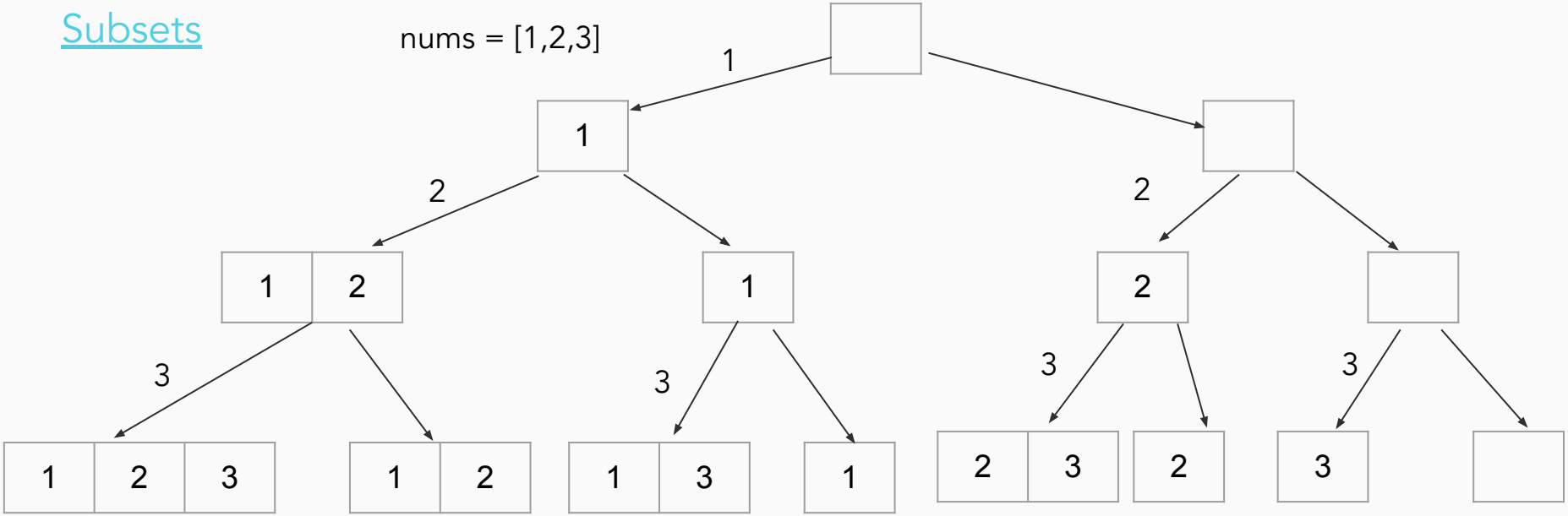
```
// let curr represent the thing you are building
// it could be an array or a combination of variables

def backtrack(curr, OTHER_ARGUMENTS...):
        if ("BASE_CASE"):
                # modify the answer
                return

        for ("ITERATE_OVER_INPUT"):
                # modify the current state
                backtrack(curr, OTHER_ARGUMENTS...)
                # undo the modification of the current state
```

Demo

Subsets



For each node, there are two different paths we can take.
Either add nums[i] or skip it

Questions?

Let’s practice!

- Review
  - Generate Parentheses
  - Word Search
- Bonus
  - Combination Sum
  - N-Queens