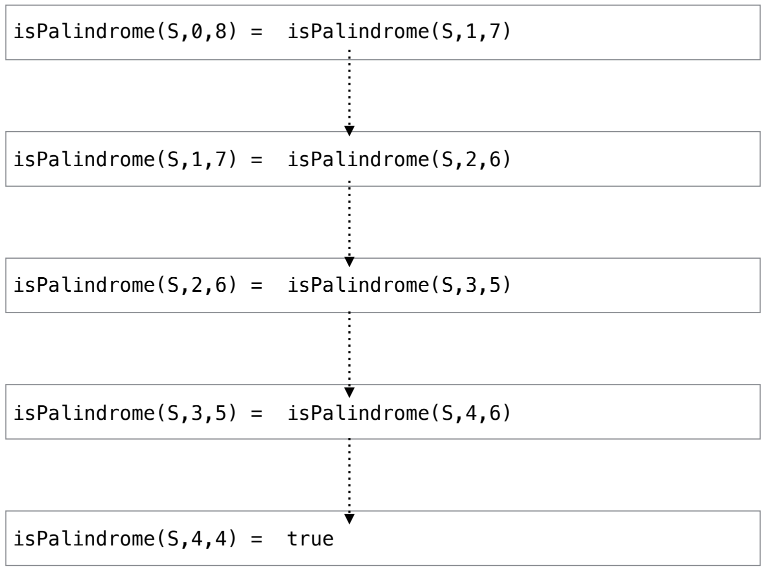
Backtracking

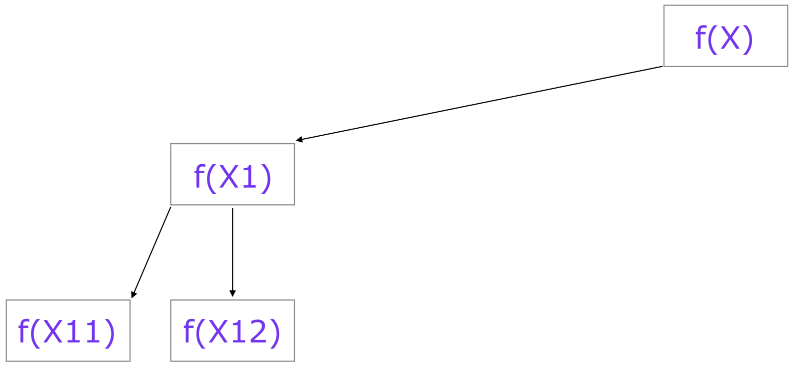


```
isPalindrome(S,0,8) =  isPalindrome(S,1,7)
isPalindrome(S,1,7) =  isPalindrome(S,2,6)
isPalindrome(S,2,6) =  isPalindrome(S,3,5)
isPalindrome(S,3,5) =  isPalindrome(S,4,6)
isPalindrome(S,4,4) =  true
```

f(X)

$$\begin{tikzcd} & f(X) \arrow[dl] \\ f(X1) & \end{tikzcd}$$

$$\begin{tikzcd}
 & & \boxed{f(X)} \arrow[dl] \\
 & \boxed{f(X1)} \arrow[dl] & \\
\boxed{f(X11)} & &
\end{tikzcd}$$


f(X)
f(X1)
f(X11)
f(X12)

$$\begin{tikzcd}
& & & \boxed{f(X)} \arrow[dll] \\
& \boxed{f(X1)} \arrow[dl] \arrow[d] \arrow[dr] & & \\
\boxed{f(X11)} & \boxed{f(X12)} & \boxed{f(X13)} &
\end{tikzcd}$$

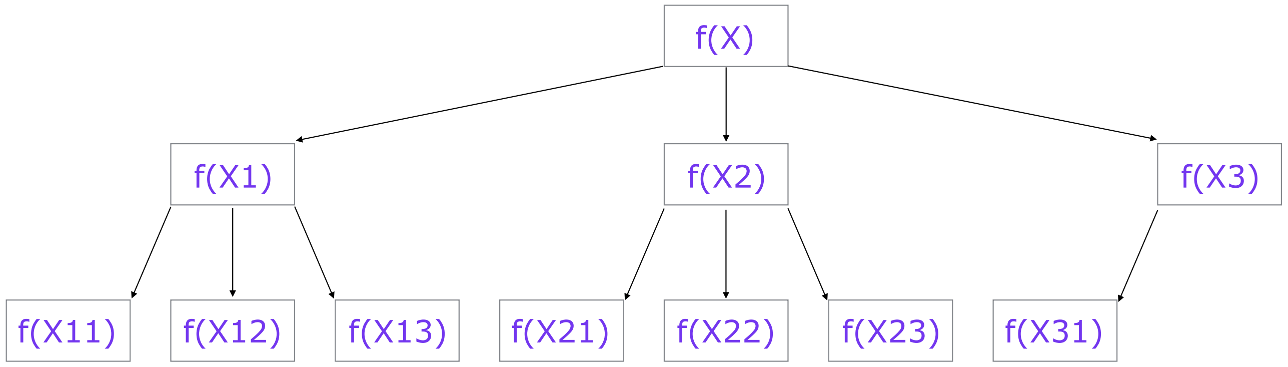
$$\begin{tikzcd}
 & & & f(X) \arrow[dll] \arrow[d] \\
 & f(X1) \arrow[dl] \arrow[d] \arrow[dr] & & f(X2) \\
f(X11) & f(X12) & f(X13) &
\end{tikzcd}$$

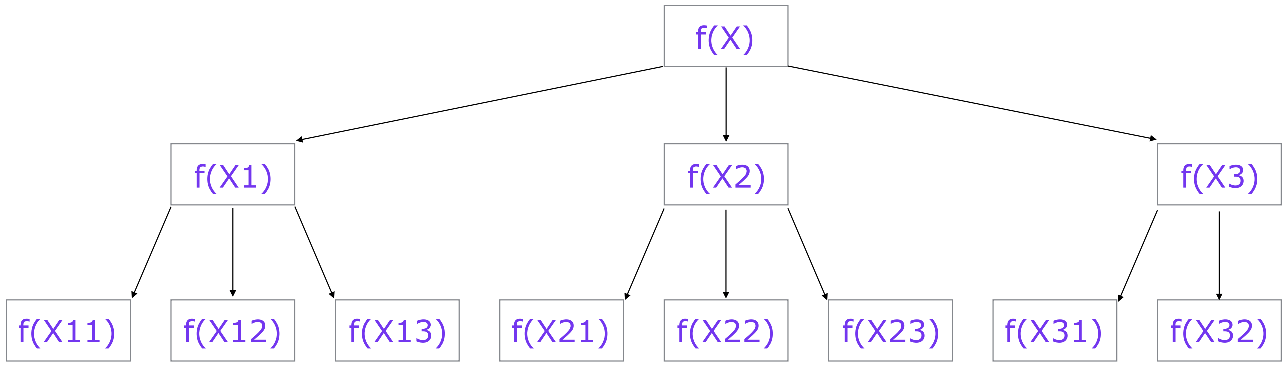
$$\begin{tikzcd}
& & & & \boxed{f(X)} \arrow[dlll] \arrow[d] \\
& \boxed{f(X1)} \arrow[dl] \arrow[d] \arrow[dr] & & & \boxed{f(X2)} \arrow[dl] \\
\boxed{f(X11)} & \boxed{f(X12)} & \boxed{f(X13)} & \boxed{f(X21)} &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & & & f(X) \arrow[llldd, start anchor=south west] \arrow[d] \\
 & f(X1) \arrow[dl] \arrow[d] \arrow[dr] & & & f(X2) \arrow[dl] \arrow[d] \\
f(X11) & f(X12) & f(X13) & f(X21) & f(X22)
\end{tikzcd}$$

$$\begin{tikzcd}
 & & & & f(X) \arrow[dlll] \arrow[d] & \\
 & f(X1) \arrow[dl] \arrow[d] \arrow[dr] & & & f(X2) \arrow[dl] \arrow[d] \arrow[dr] & \\
f(X11) & f(X12) & f(X13) & f(X21) & f(X22) & f(X23)
\end{tikzcd}$$

$$\begin{tikzcd}
 & & & & f(X) \arrow[dlll] \arrow[d] \arrow[drrr] & & & \\
 & f(X1) \arrow[dl] \arrow[d] \arrow[dr] & & & f(X2) \arrow[dl] \arrow[d] \arrow[dr] & & & f(X3) \\
f(X11) & f(X12) & f(X13) & f(X21) & f(X22) & f(X23) & &
\end{tikzcd}$$


f(X)
f(X1)
f(X2)
f(X3)
f(X11)
f(X12)
f(X13)
f(X21)
f(X22)
f(X23)
f(X31)


f(X)
f(X1)
f(X2)
f(X3)
f(X11)
f(X12)
f(X13)
f(X21)
f(X22)
f(X23)
f(X31)
f(X32)

$$\begin{tikzcd}
 & & & & f(X) \arrow[dlll] \arrow[d] \arrow[drrr] & & & & \\
 & f(X1) \arrow[dl] \arrow[d] \arrow[dr] & & & f(X2) \arrow[dl] \arrow[d] \arrow[dr] & & & f(X3) \arrow[dl] \arrow[d] \arrow[dr] & \\
f(X11) & f(X12) & f(X13) & f(X21) & f(X22) & f(X23) & f(X31) & f(X32) & f(X33)
\end{tikzcd}$$

- Every recursive call represents a decision or choice at each step. Calls itself once for each of the choice.
- Try out all the paths and pick those which meets the criteria.
- These problems are called Exhaustive Search or Combinatorial Search Problems.

Examples

1. Given an array of integers, print all the permutations of the given array.

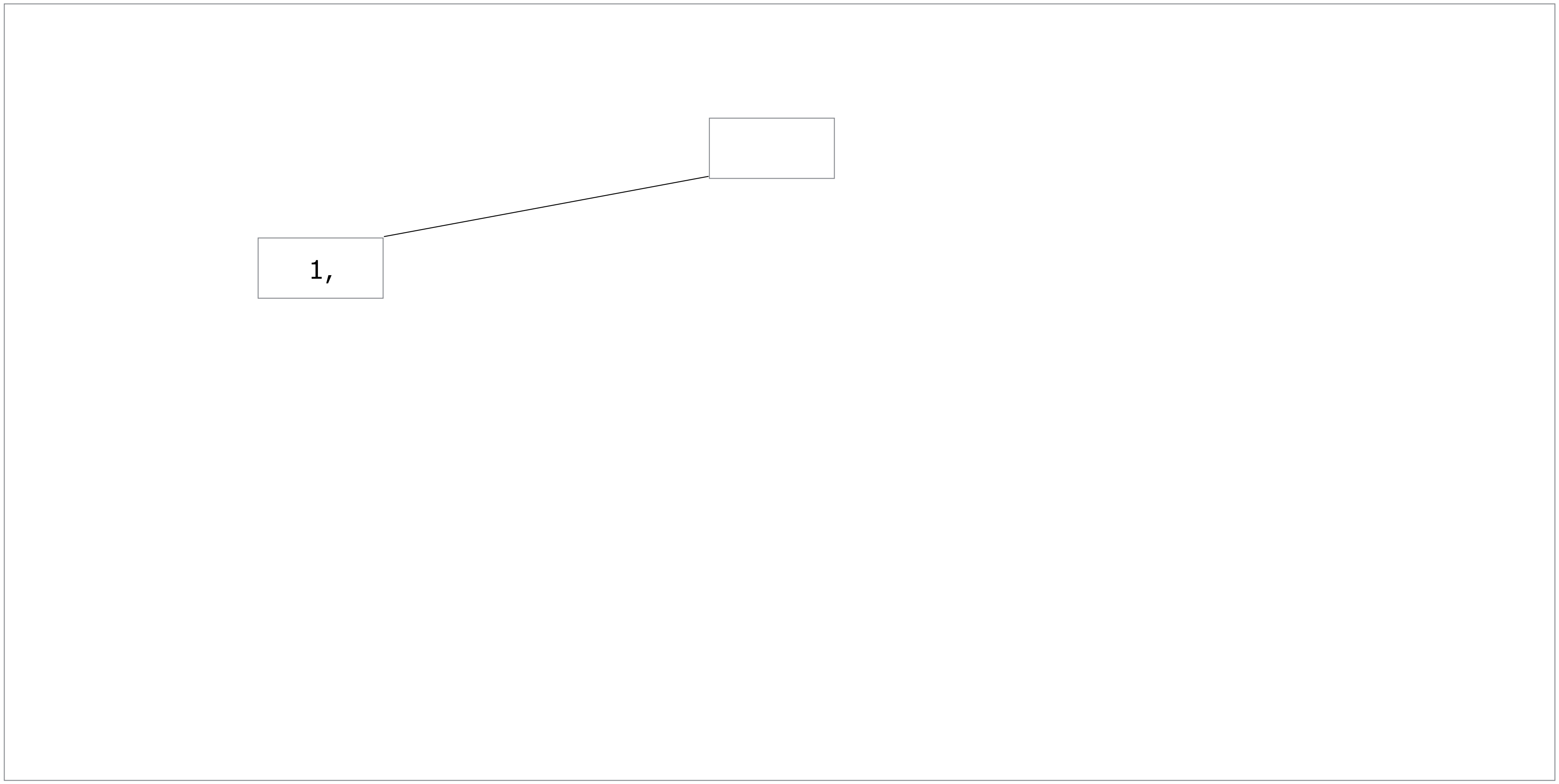
Input=[1,2,3]

Output:

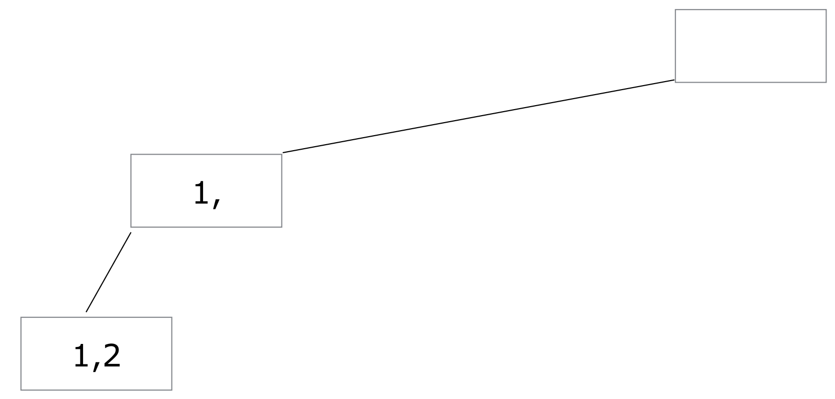
```
1,2,3

1,3,2

2,1,3

2,3,1

3,1,2

3,2,1
```

Backtracking

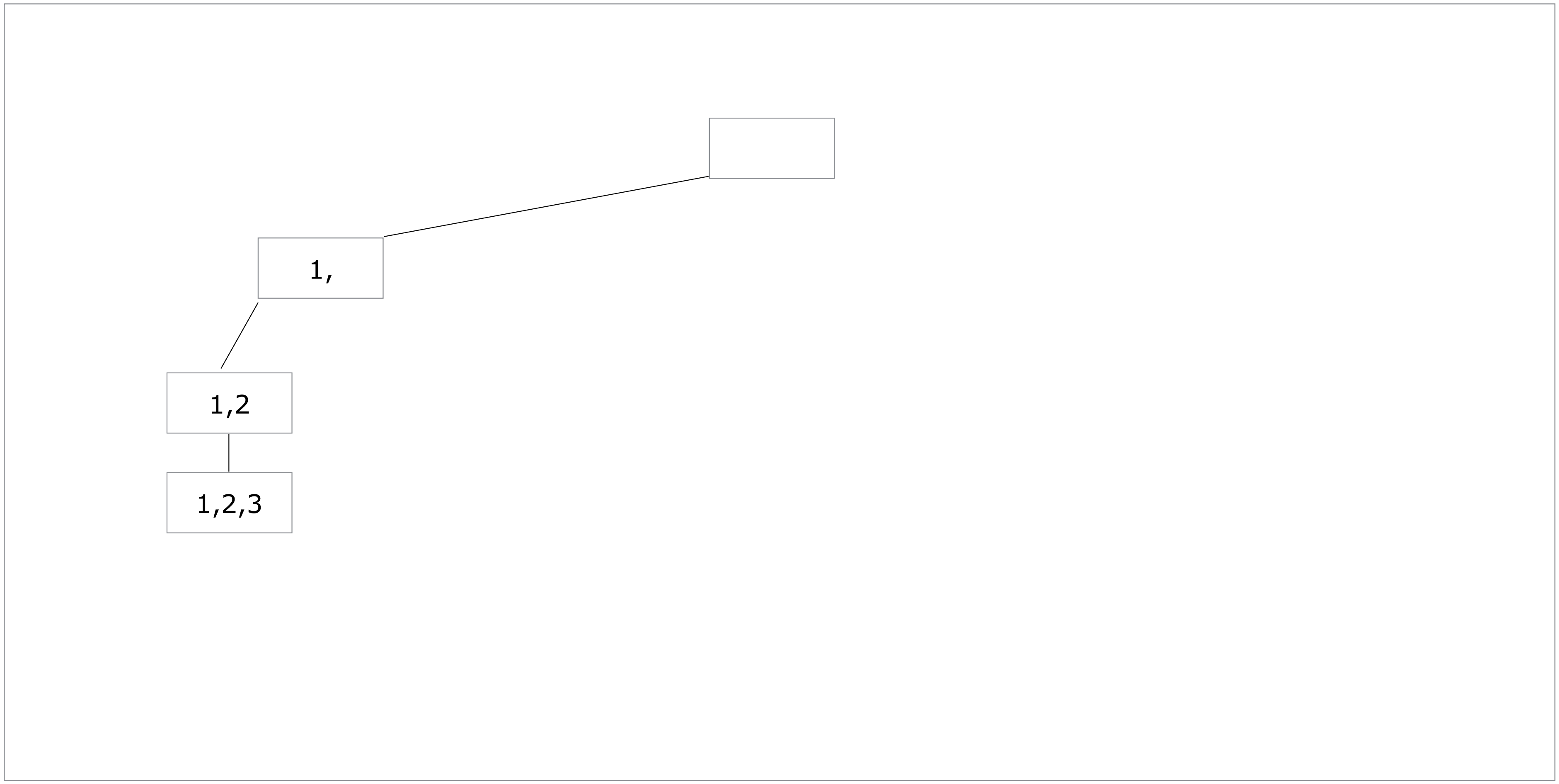
Backtracking



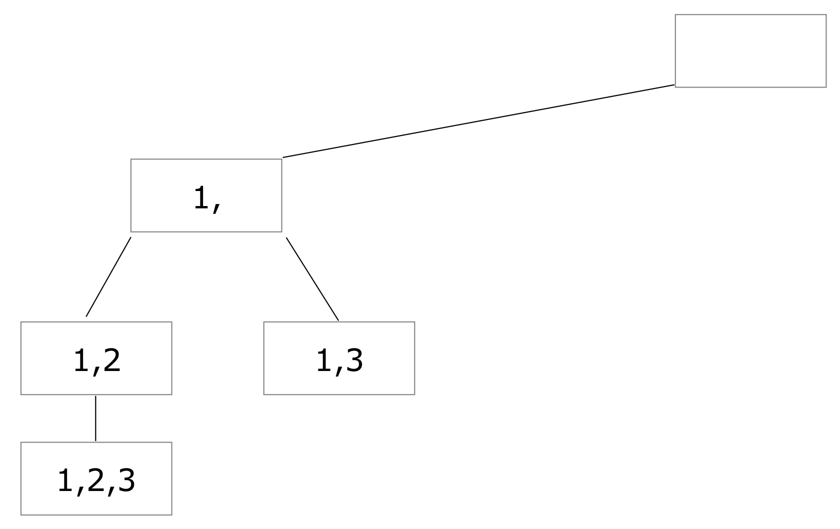
Backtracking



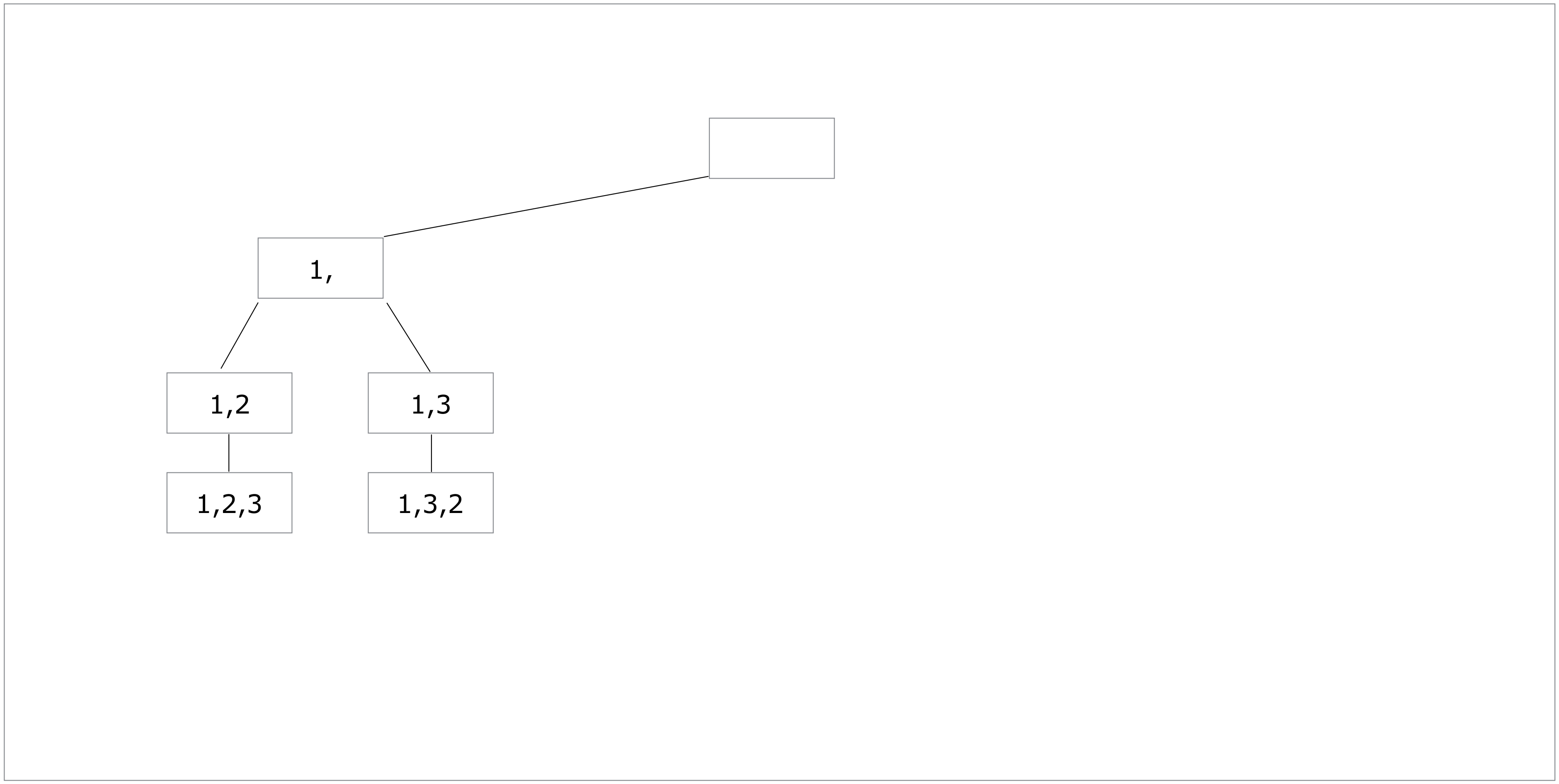
Backtracking



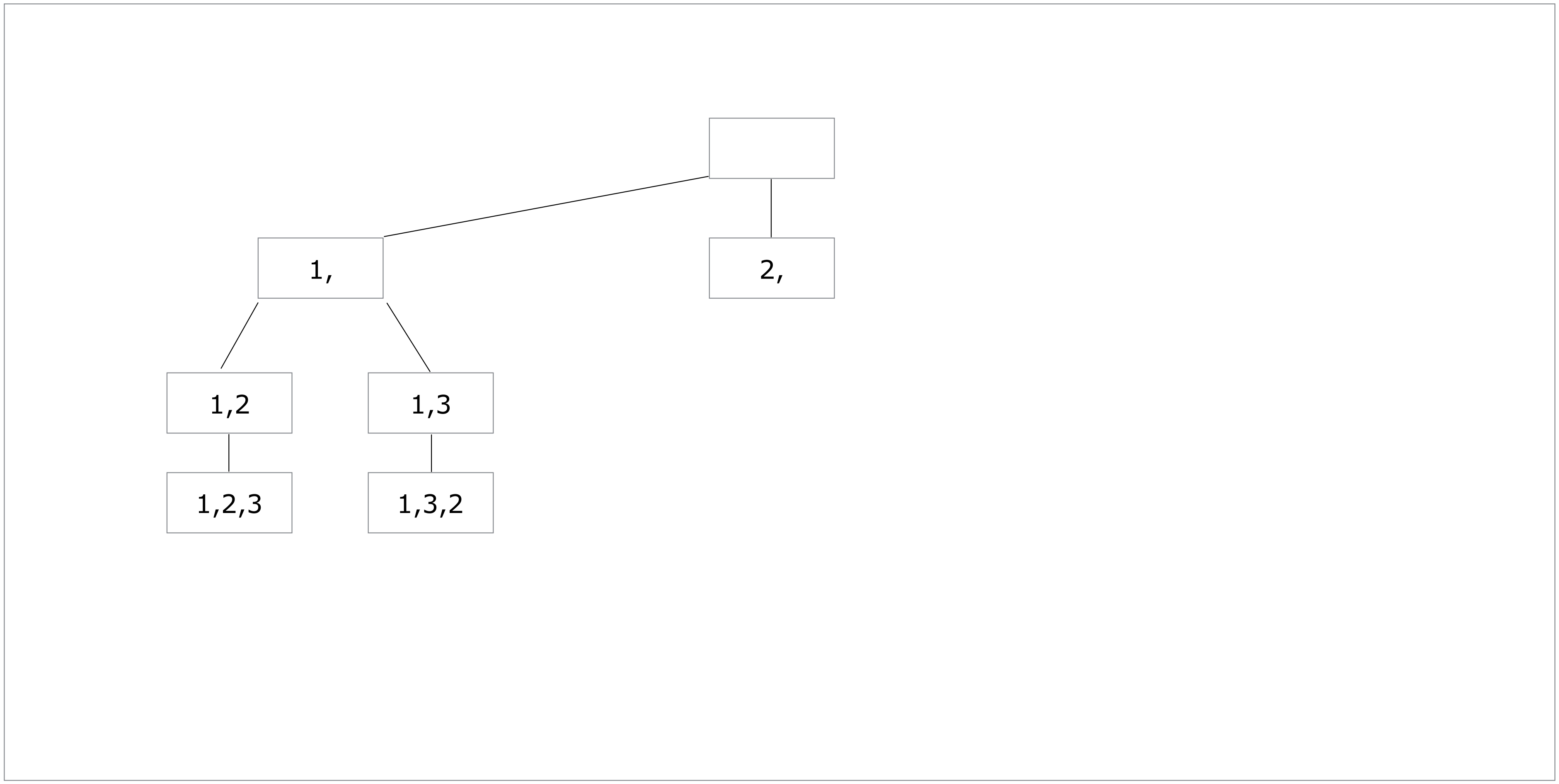
Backtracking



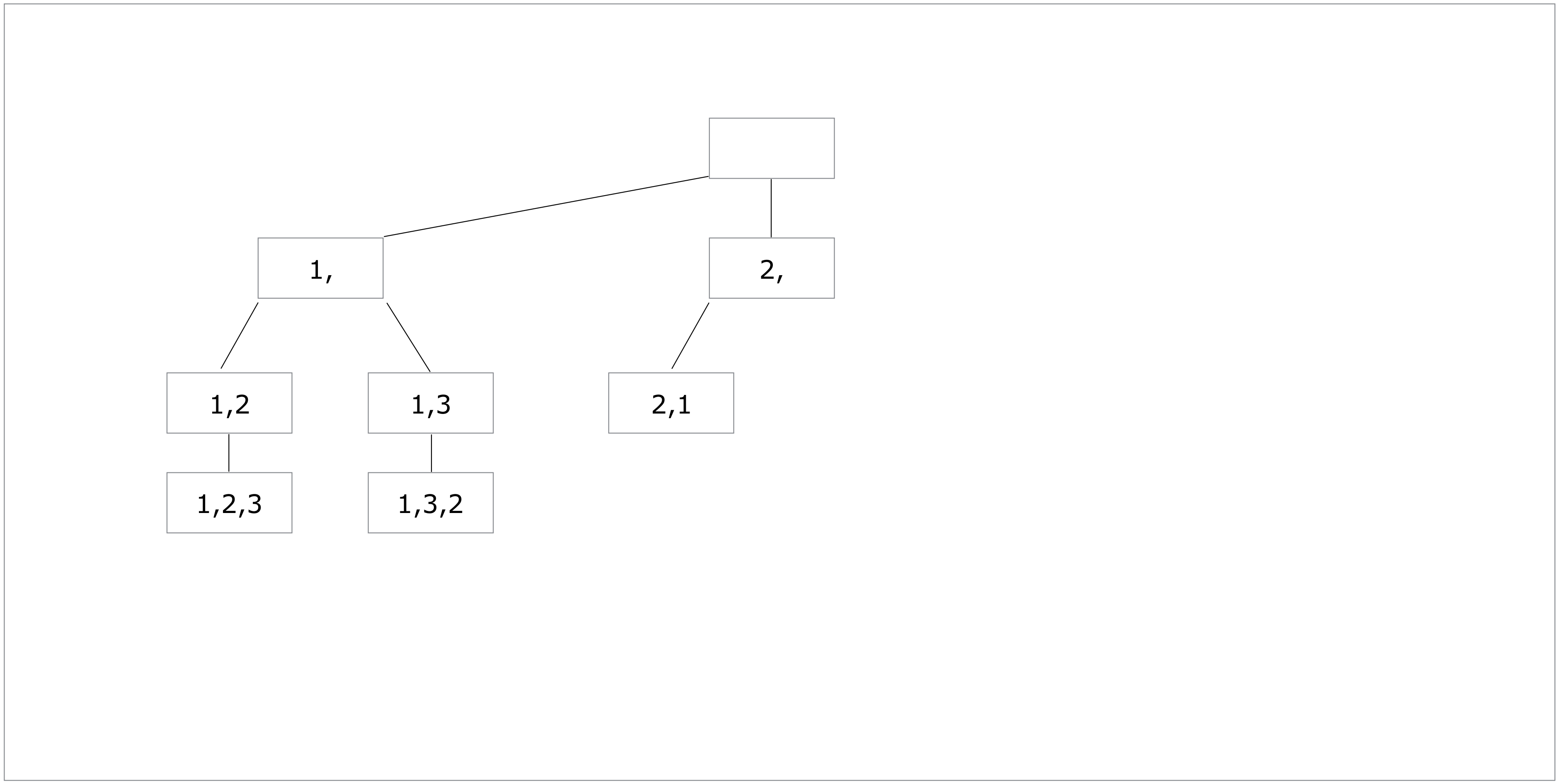
Backtracking



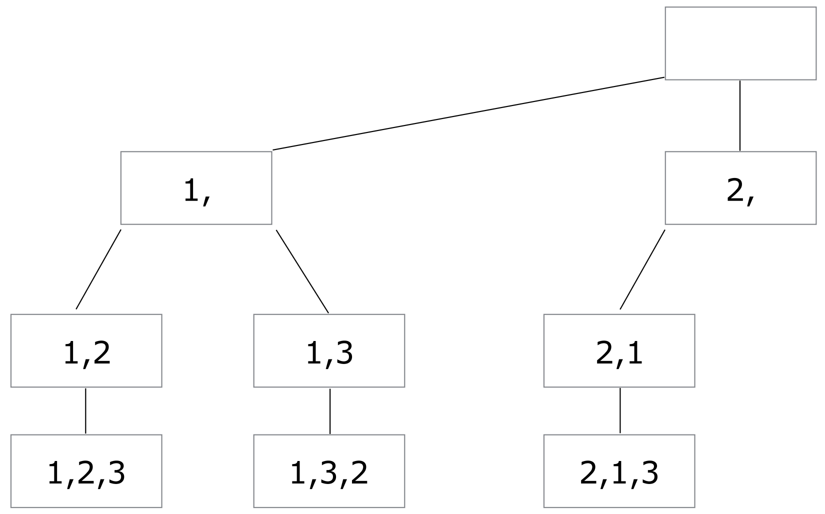
Backtracking



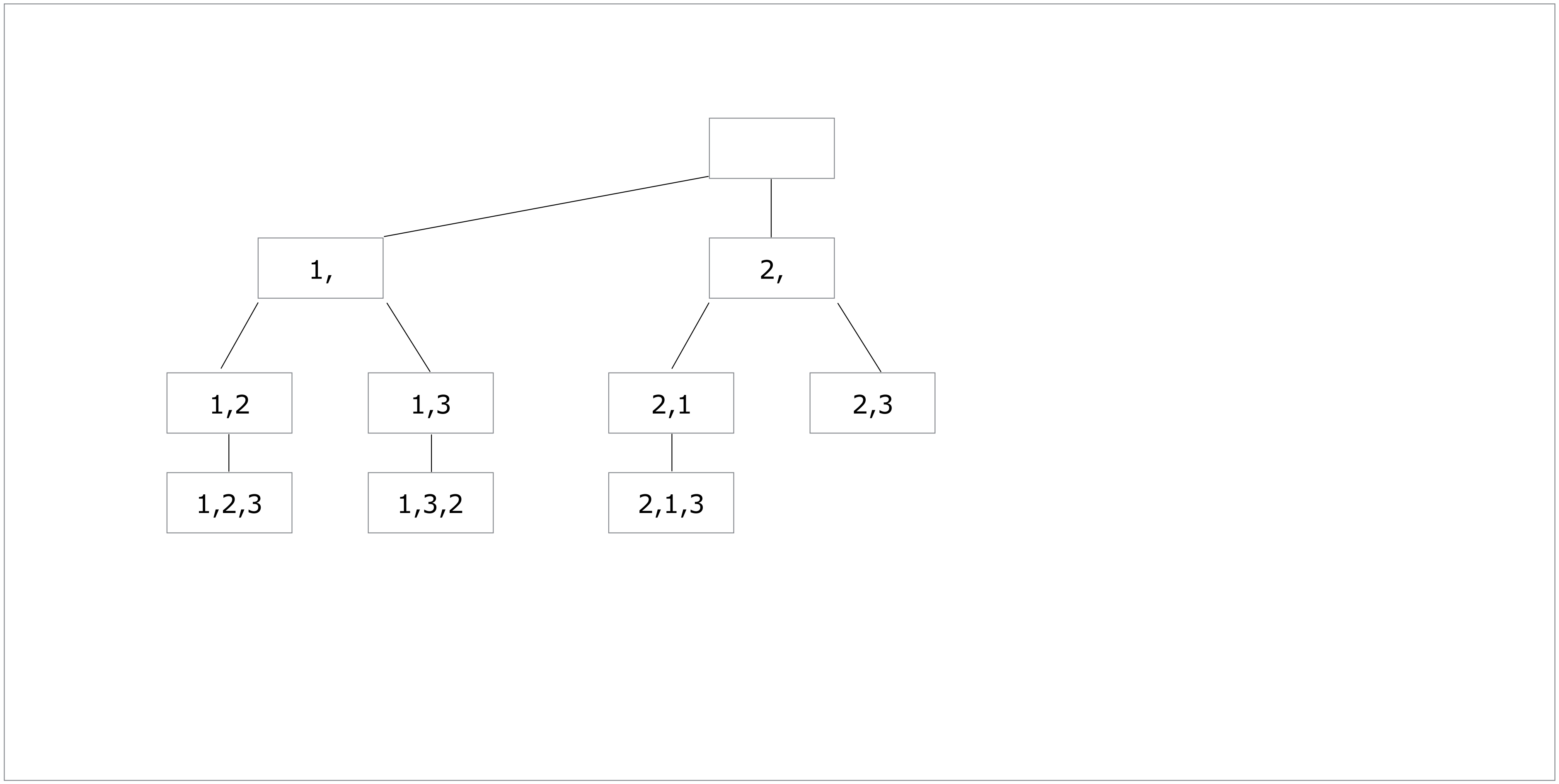
Backtracking



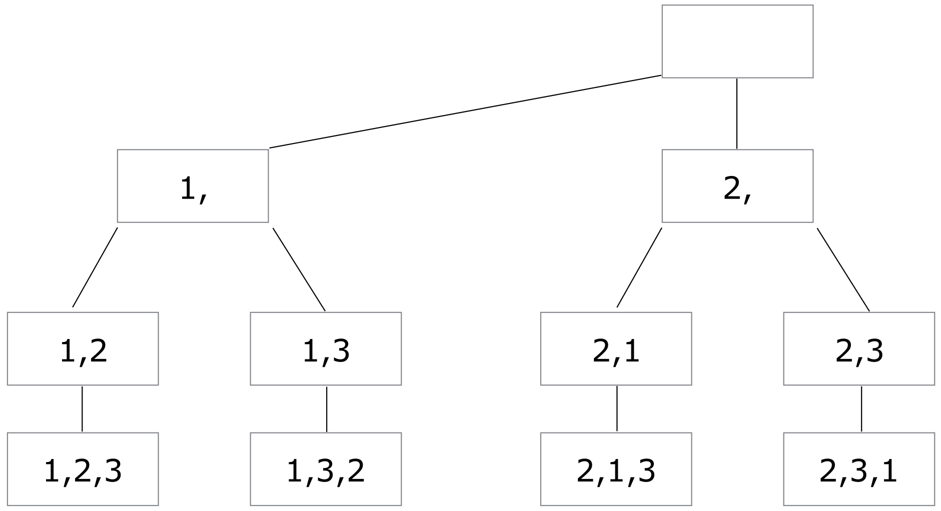
Backtracking



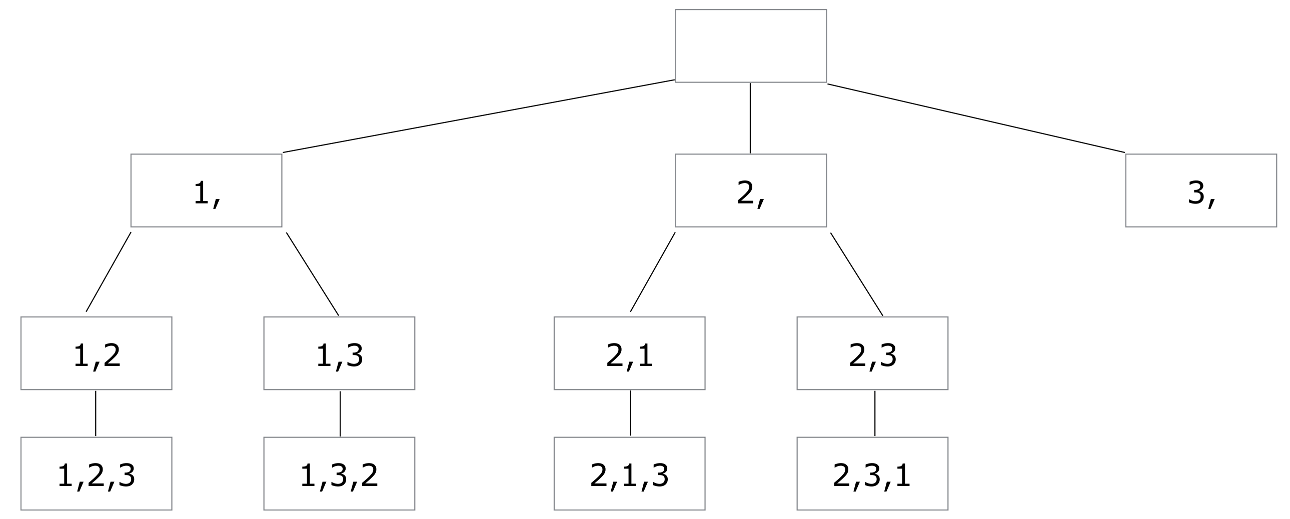
Backtracking



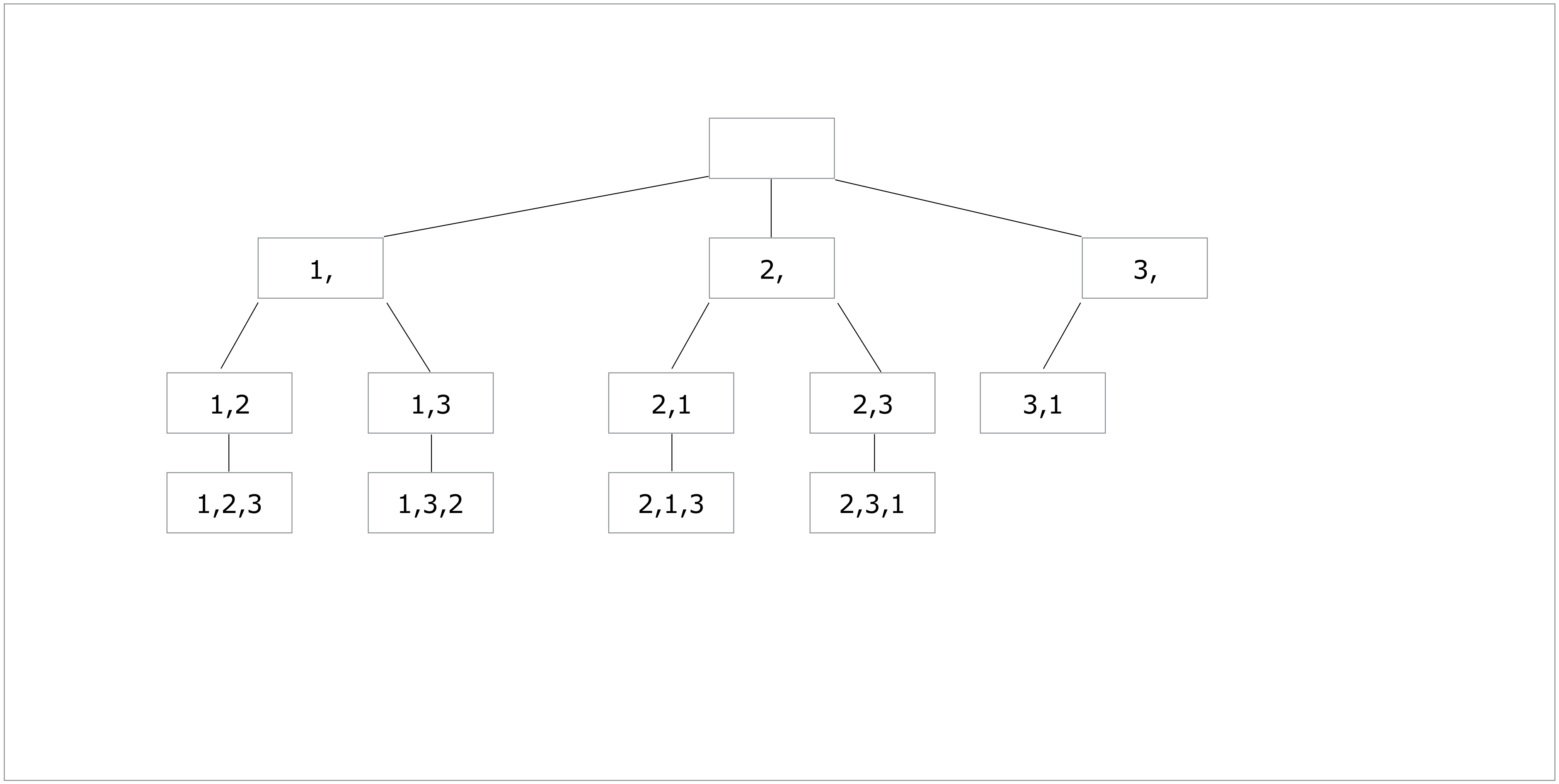
Backtracking



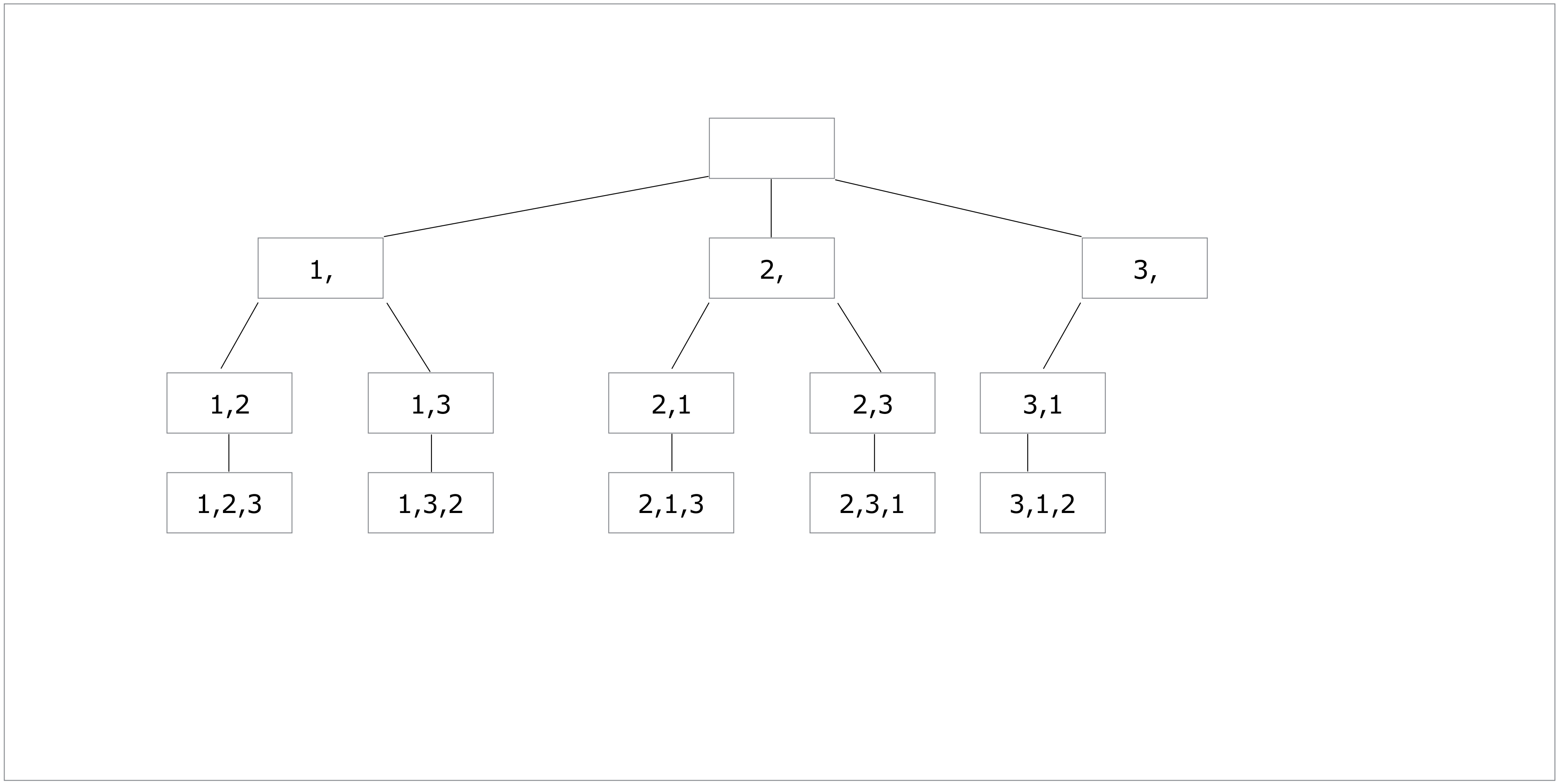
Backtracking



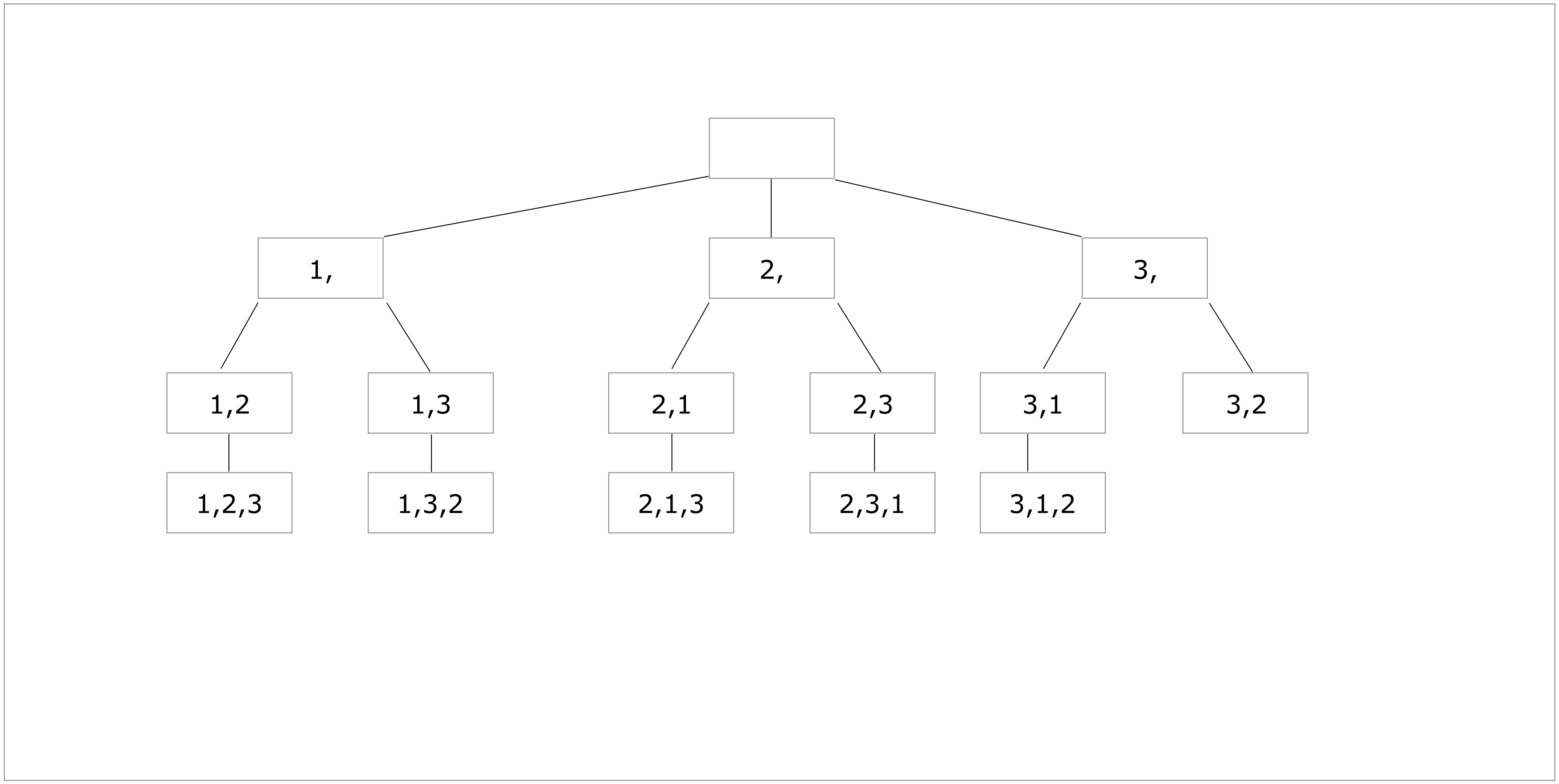
Backtracking



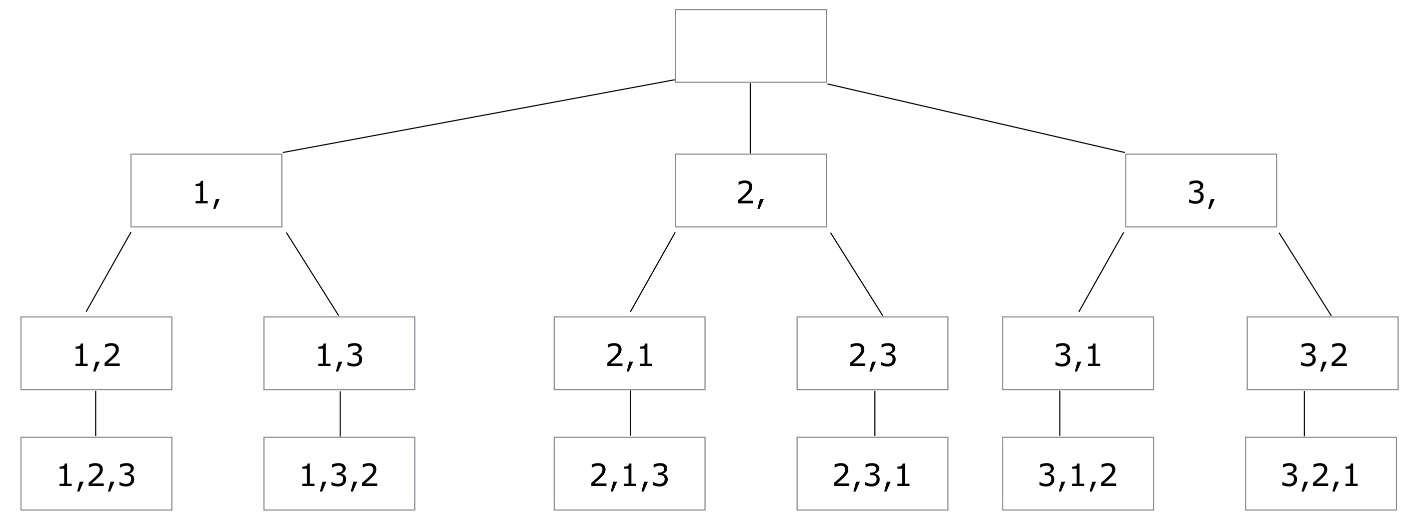
Backtracking



Backtracking



Backtracking



What is Backtracking ?

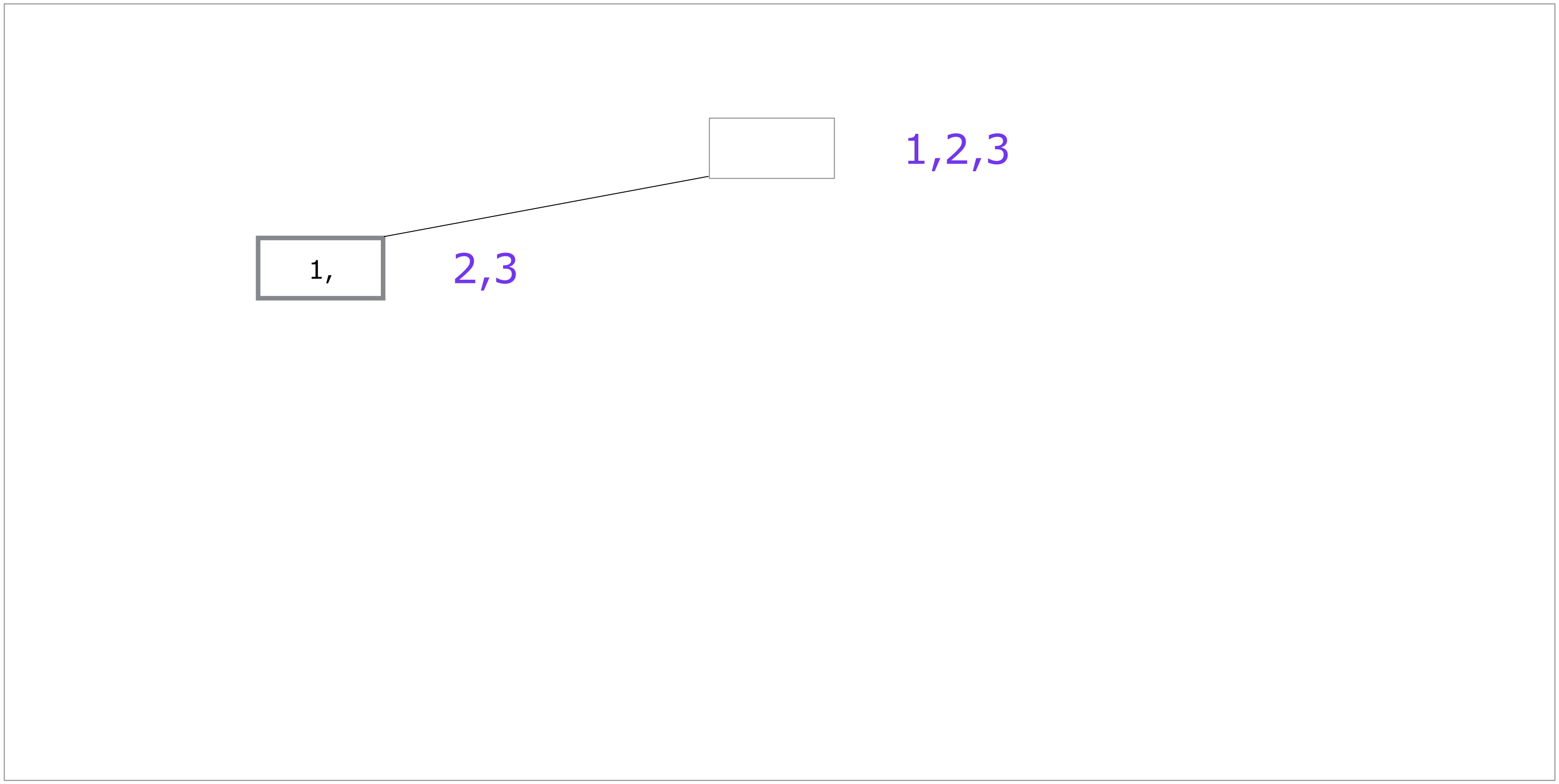
Backtracking is a systematic approach to generate all the possibilities for a given exhaustive search/combinatorial search problem using recursion.

We start off with initial state with an empty solution, at every step we extend the partial solution by considering all the possibilities.

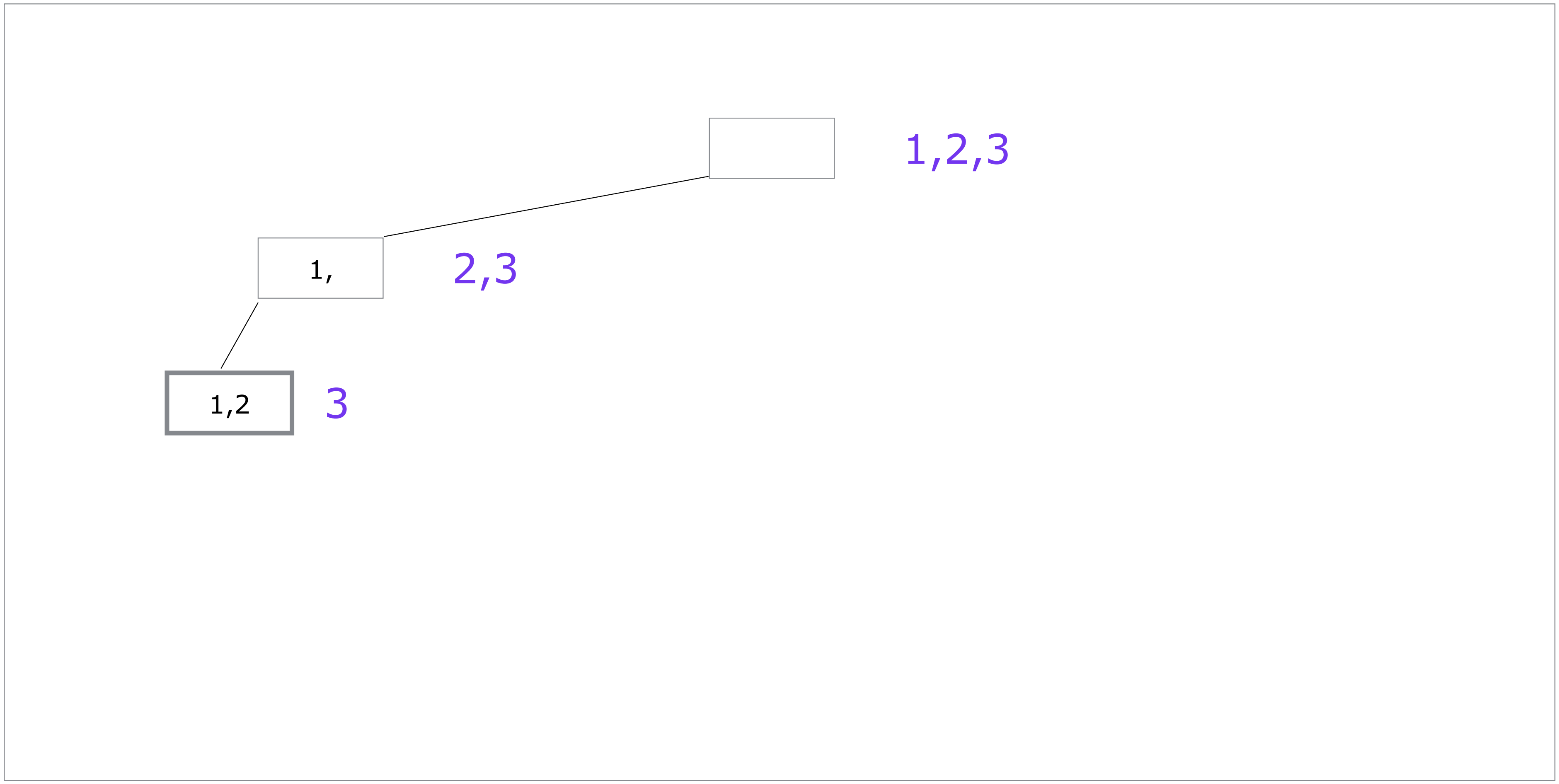
We check if the partial solution meets the given criteria, if it does not then we backtrack one step and try out another possibility.

Backtracking

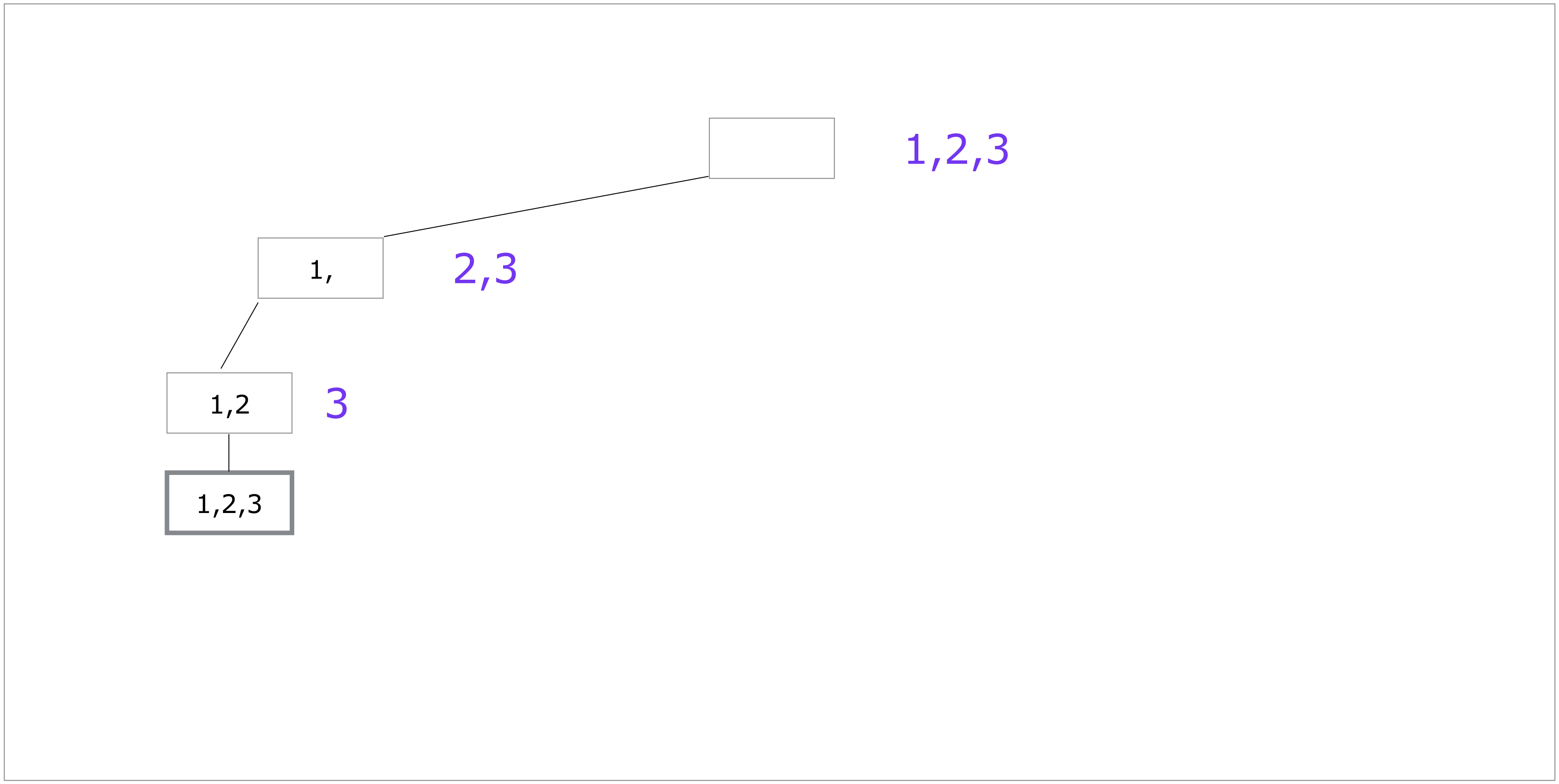
Backtracking



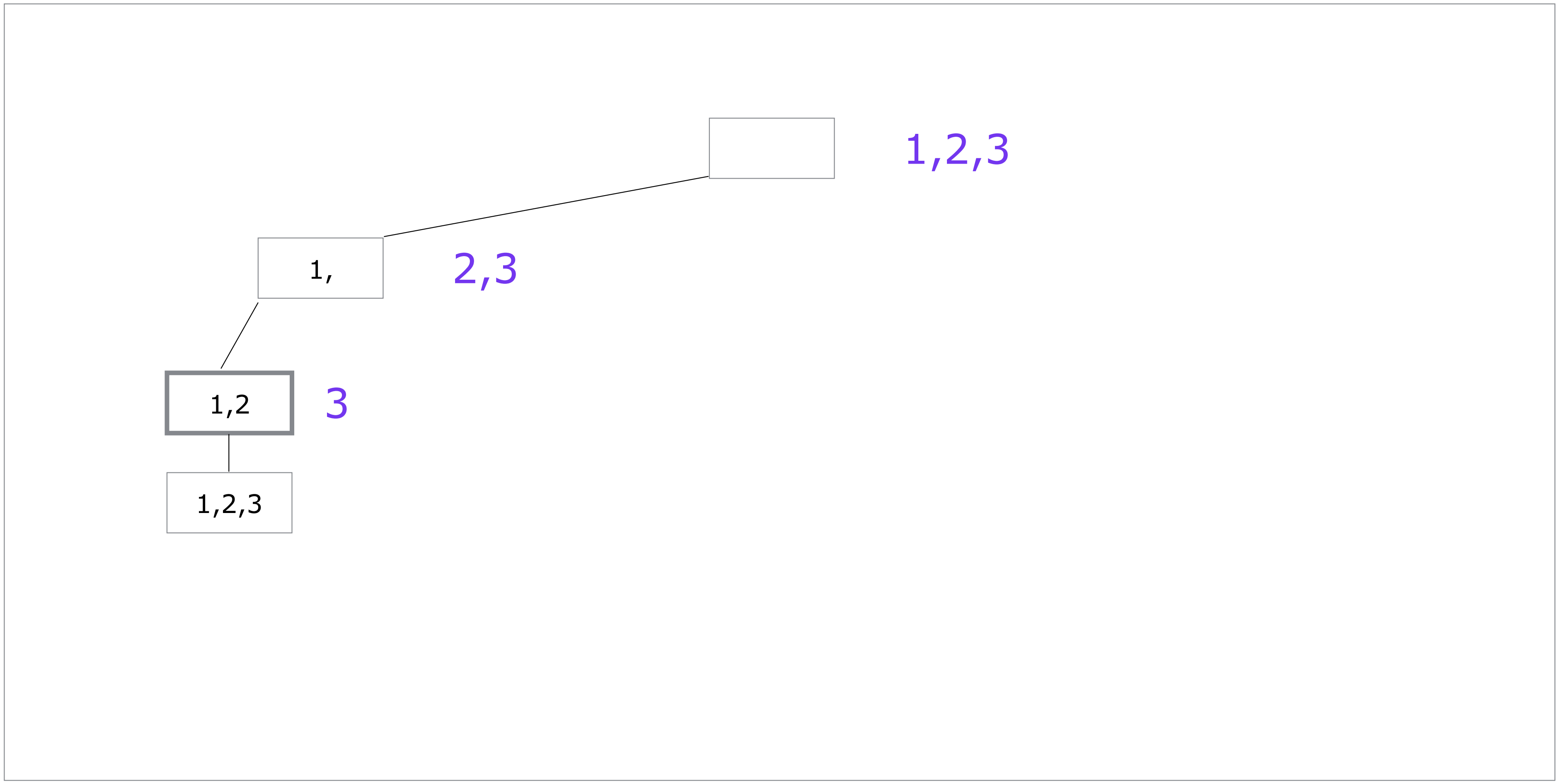
Backtracking



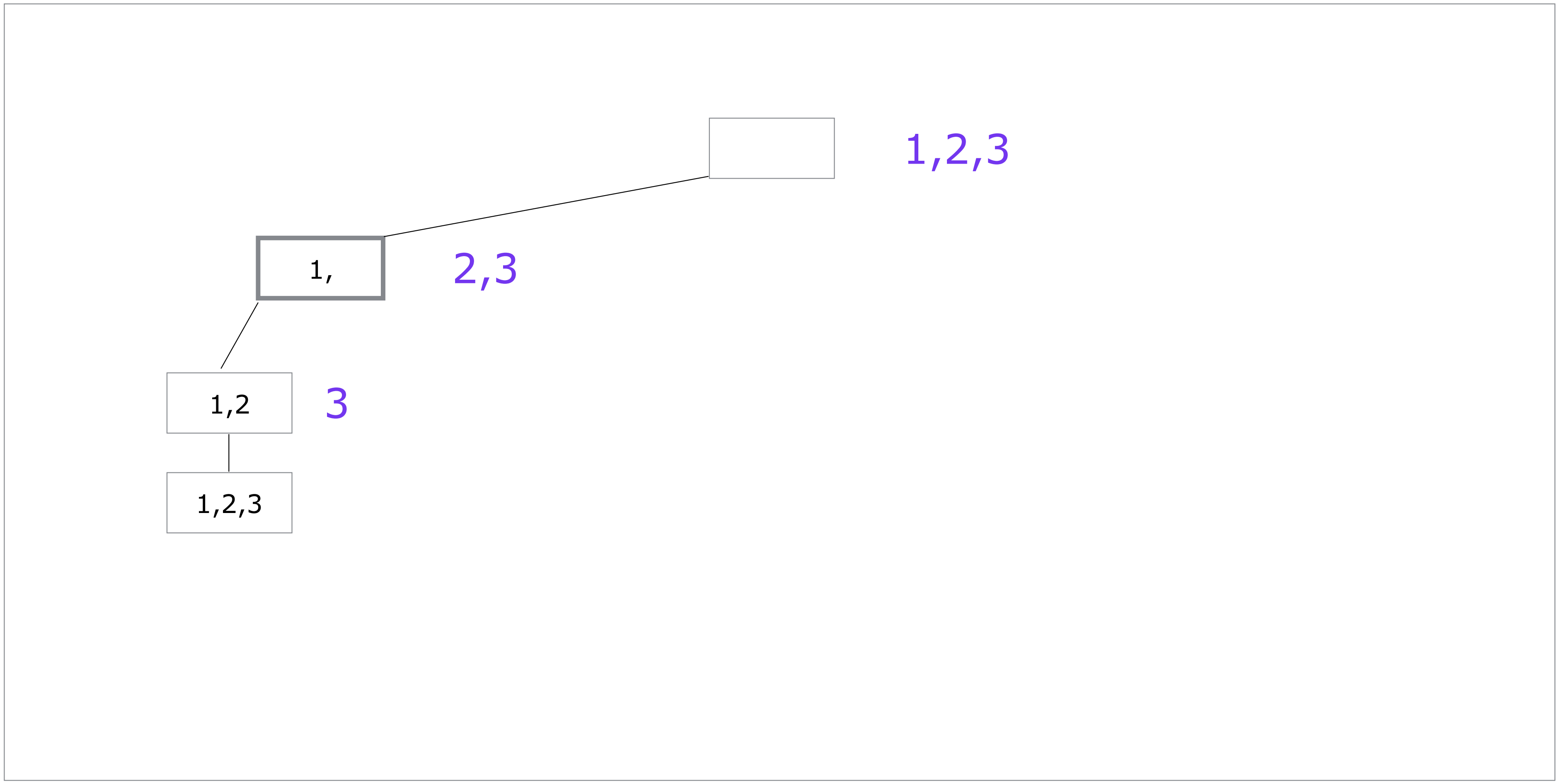
Backtracking



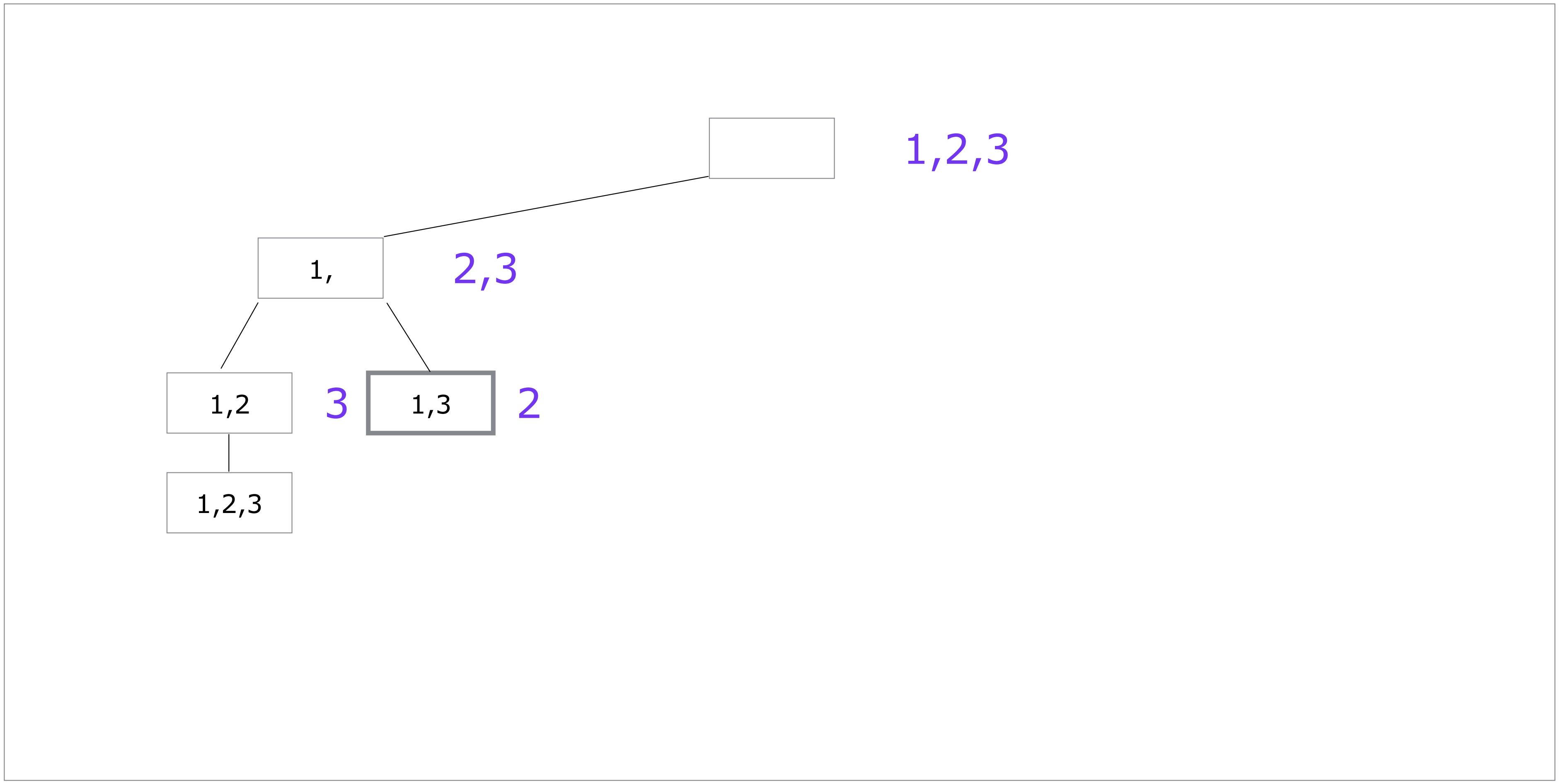
Backtracking



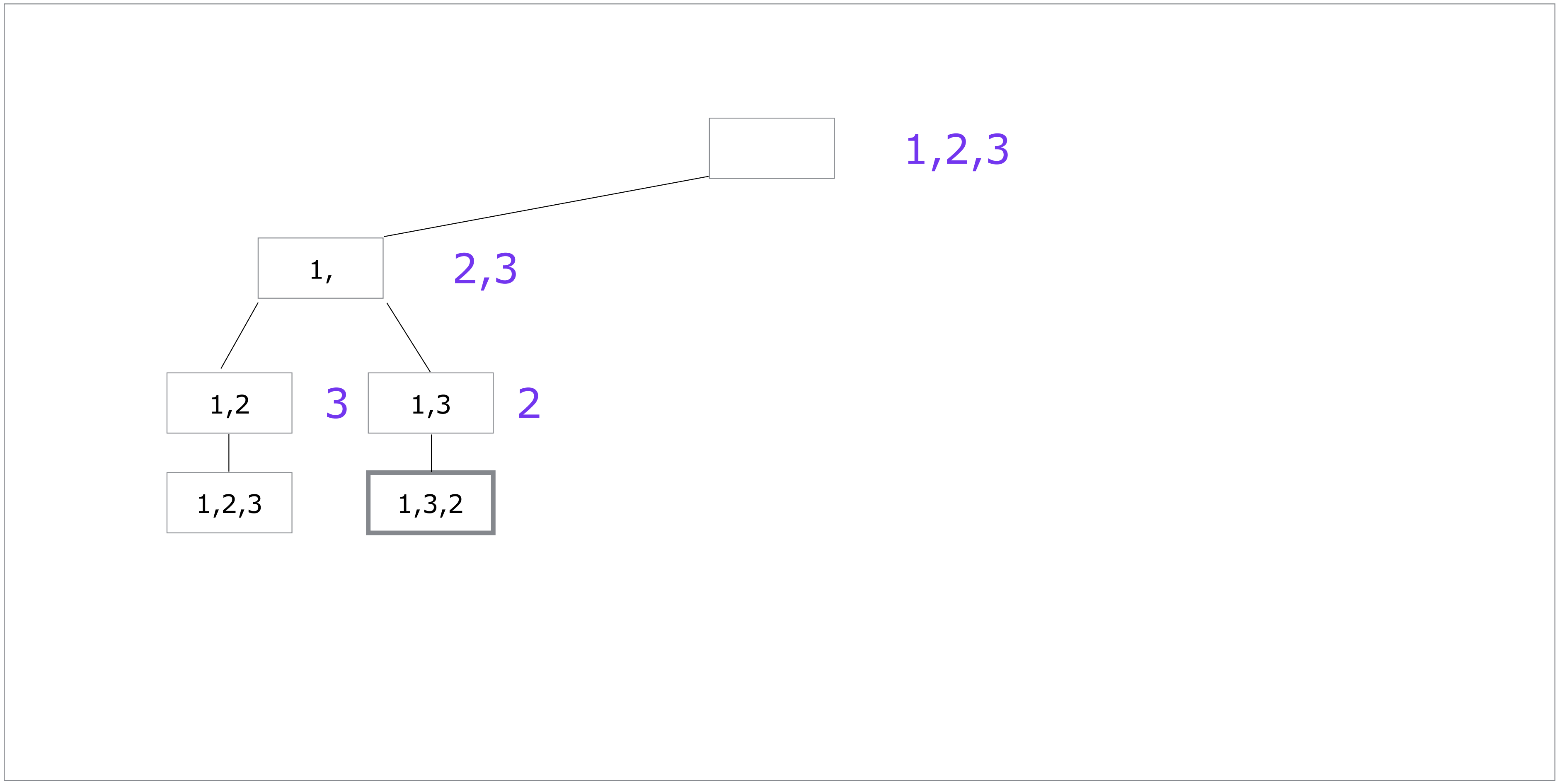
Backtracking



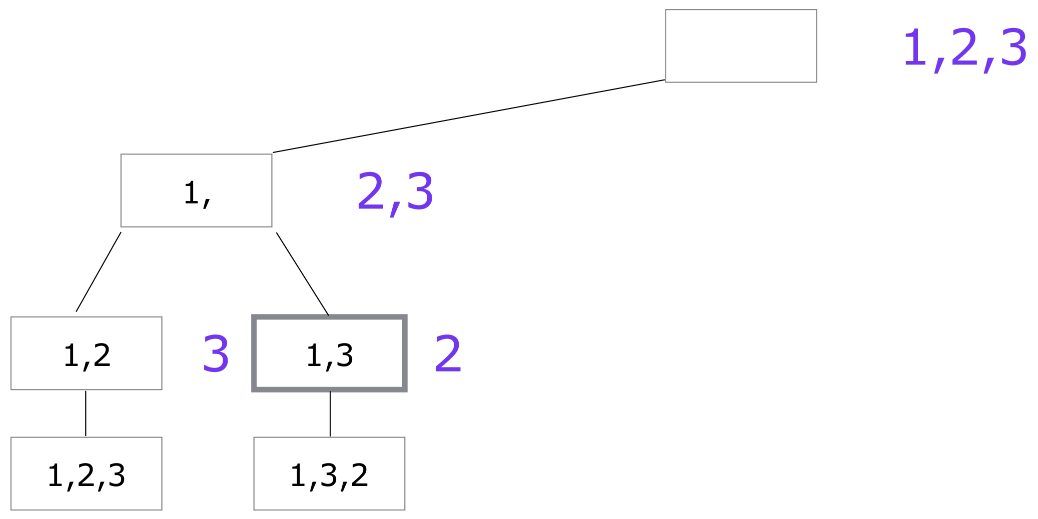
Backtracking



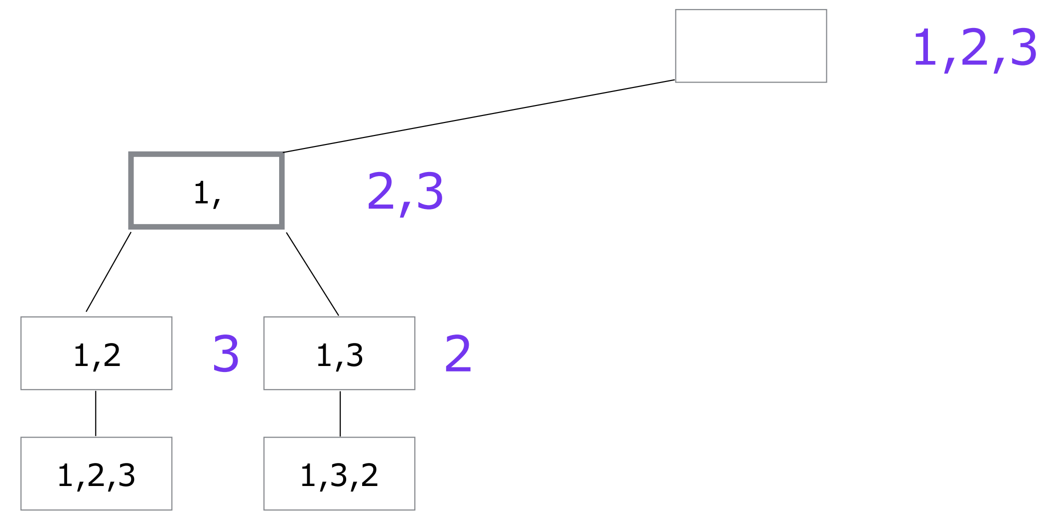
Backtracking



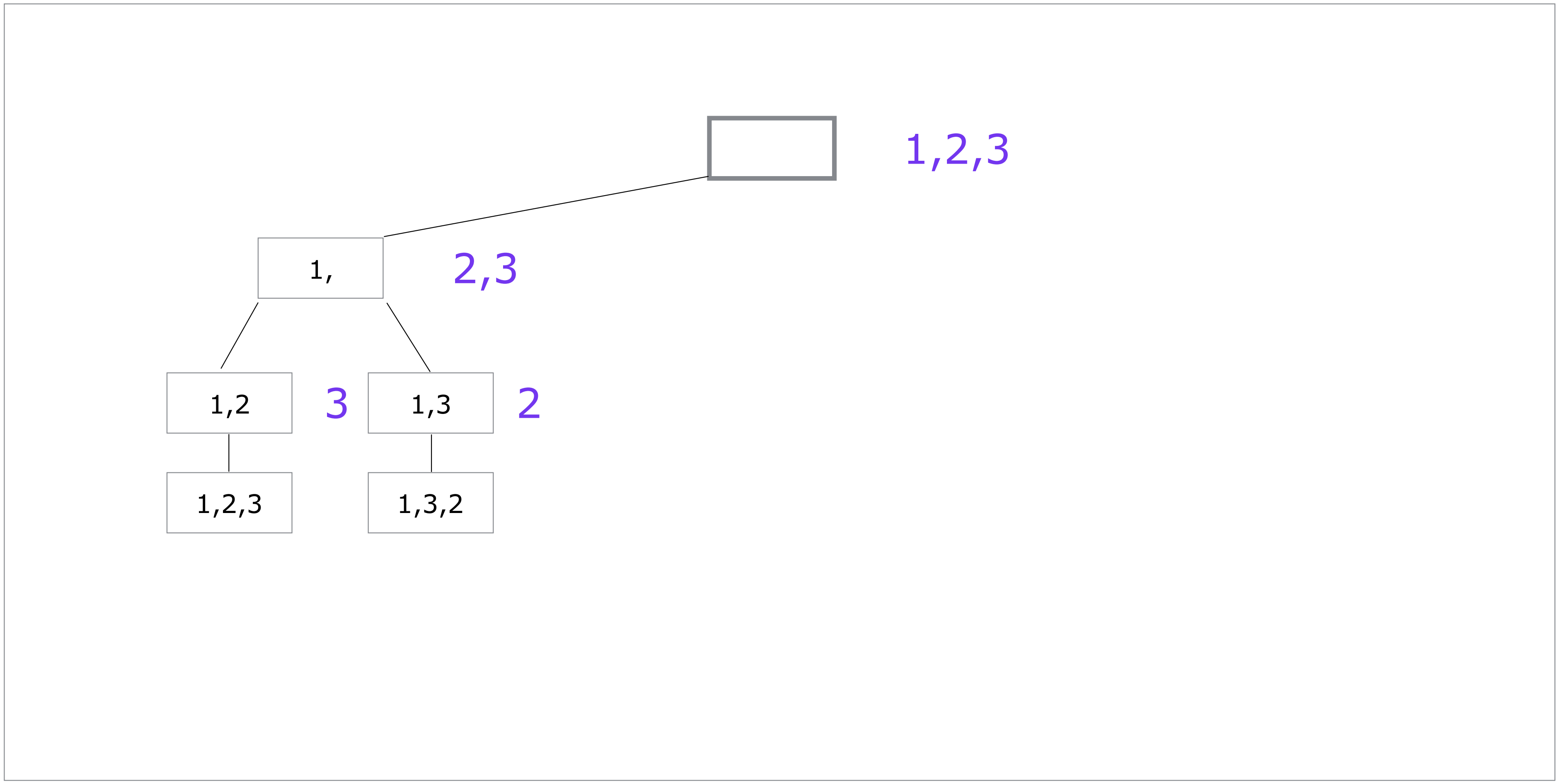
Backtracking



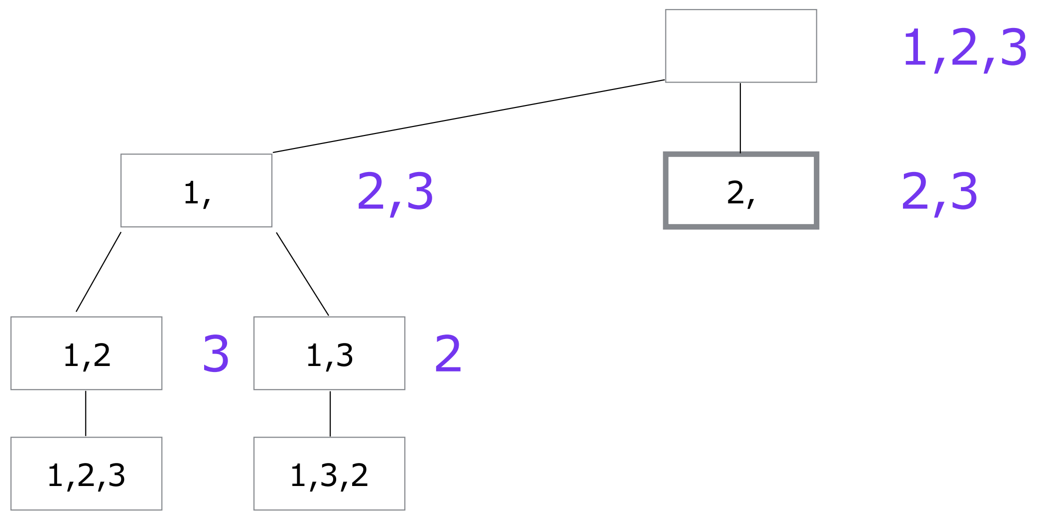
Backtracking



Backtracking



Backtracking



Pseudo code

```
function(input,partial,output…){
  if isValidSolution(partial){
      processSolution(partial)
        return
  }
  candidates = generateCandidates(input,partial)
  for c in candidate {
      addCandidate(c,input,partial)
      function(input,partial,output)
      removeCandidate(c,input,partial)
  }
}
```

Java

```java
public static void permutation(int[] input, ArrayList<Integer>
partial, boolean[] used) {
    if (partial.size() == input.length) {
        System.out.println(Arrays.toString(partial.toArray()));
        return;
    }
    for (int i = 0; i < input.length; i++) {
        if (!used[i]) {
            used[i] = true;
            partial.add(input[i]);
            permutation(input, partial, used);
            used[i] = false;
            partial.remove(partial.size() - 1);
        }
    }
}
```

Python

```python
def permutation(input_list, partial, used):
    if len(partial) == len(input_list):
        print(partial)
    else:
        for i in range(0, len(input_list)):
            if not used[i] and not (input_list[i] ==
input_list[i - 1] and not used[i - 1]):
                used[i] = True
                partial.append(input_list[i])
                permutation(input_list, partial, used)
                used[i] = False
                partial.pop(len(partial) - 1)
```

Why are we talking about these problems, how is it related to Dynamic Programming ?

- Dynamic programming is mainly used to solve optimization problems where at every step you have multiple choices and we have to make decisions at every step which gives us most optimal solution.
- To come up with Dynamic Programming solution from scratch , the first step is to come up with a naive recursive solution. So you should know how to use recursion to generate all possible solutions.

These problems can be broadly divided into two kinds,

1. To generate permutation

2. To generate combinations

In permutation the order matters and in combination the order does not matter.

Hence the name Combinatorial search

2. Given an input array and an integer ‘K’ which is atmost the size of the array, generate all the ways we can choose K integers from the array.

Example

Input = [3,2,5,8] , K=3

[3, 2, 5]

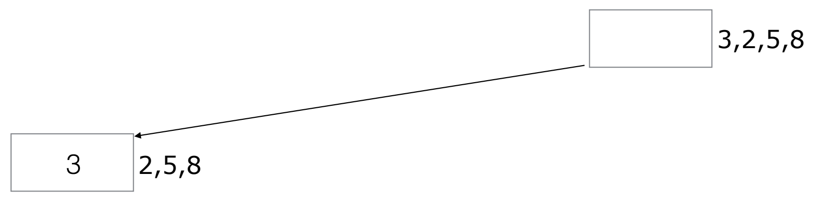
[3, 2, 8]

[3, 5, 8]

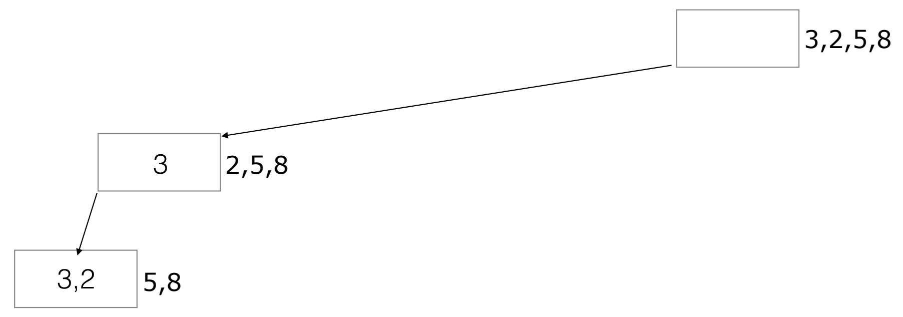
[2, 5, 8]

Backtracking

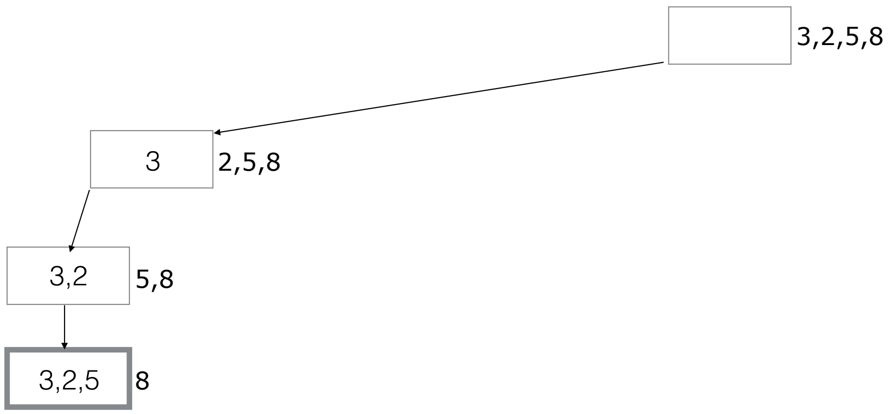
Backtracking



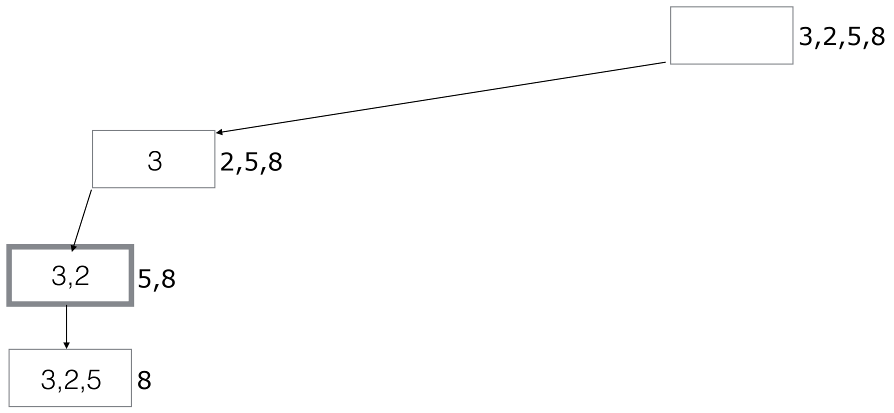
Backtracking



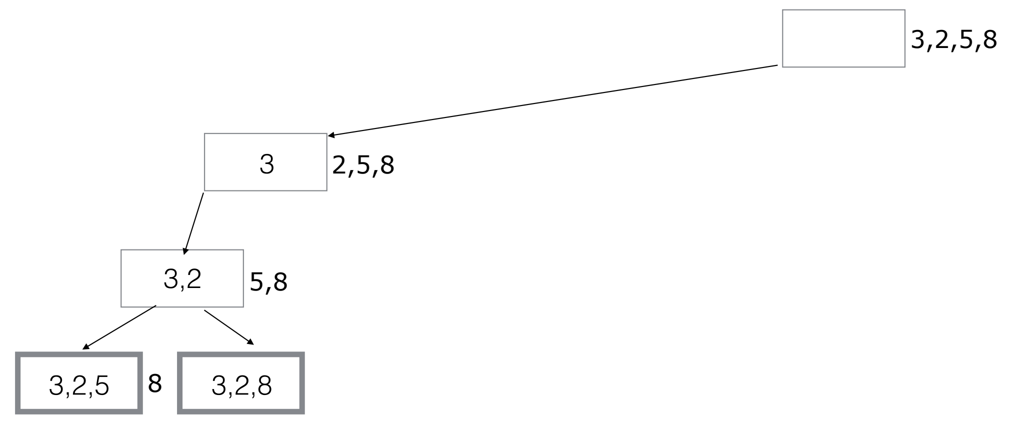
Backtracking



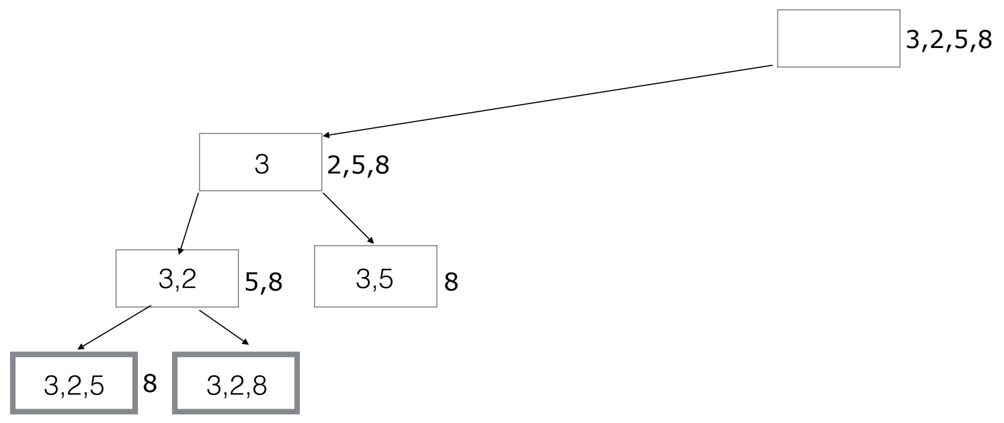
Backtracking



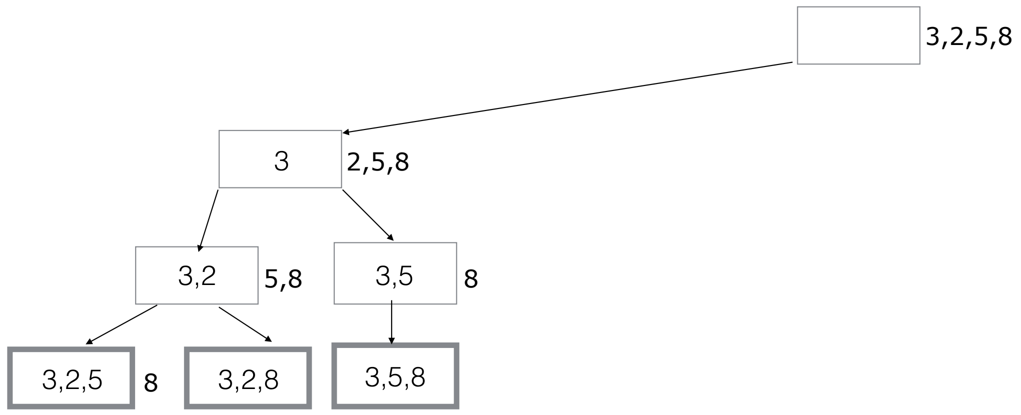
Backtracking



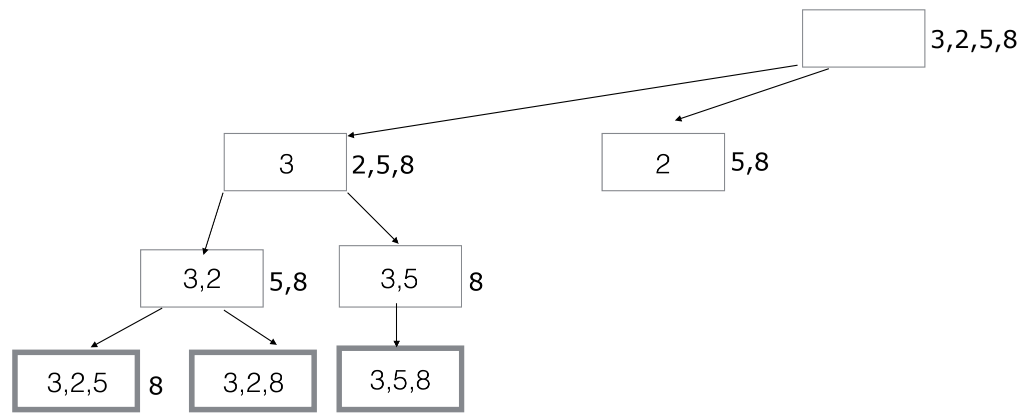
Backtracking



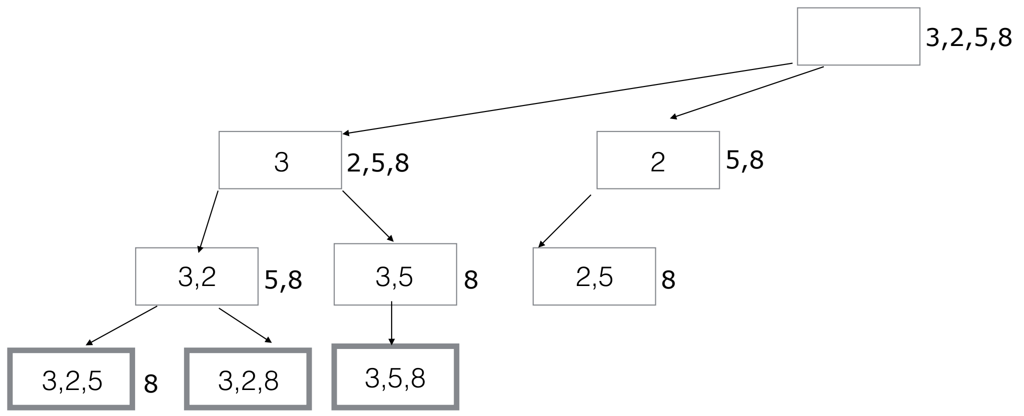
Backtracking



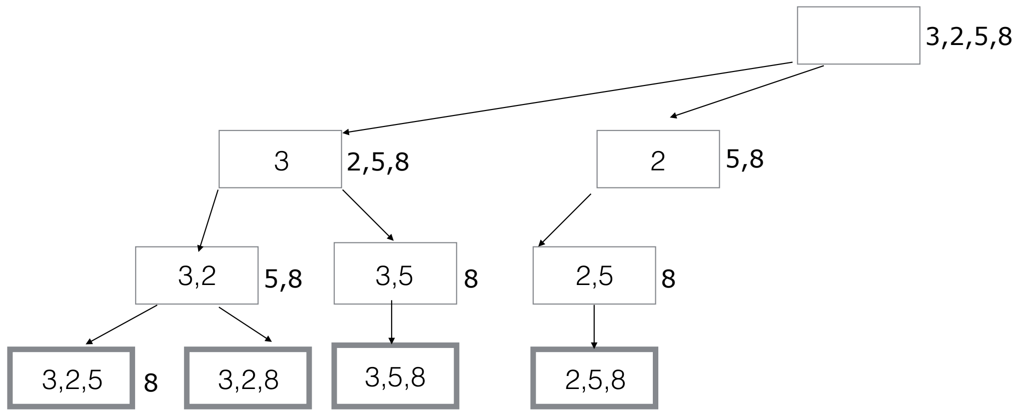
Backtracking



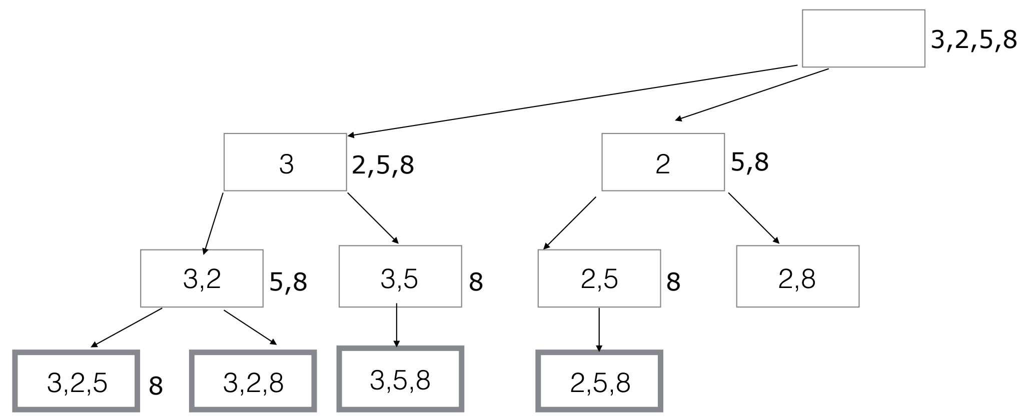
Backtracking



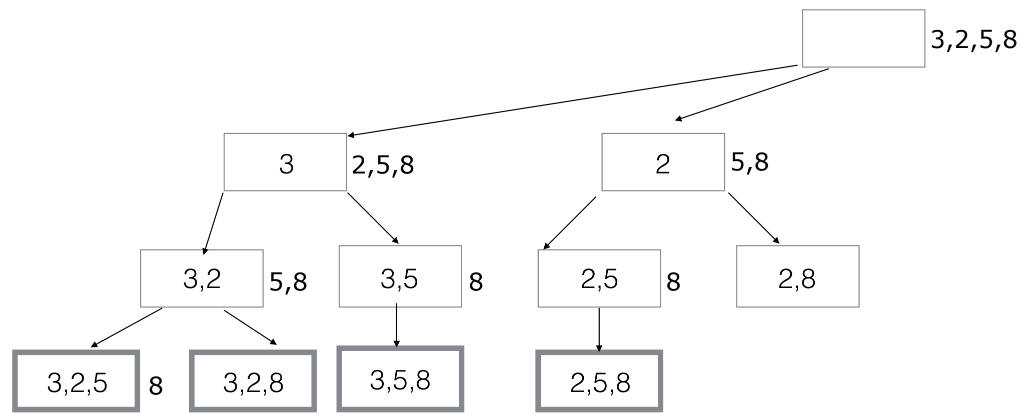
Backtracking



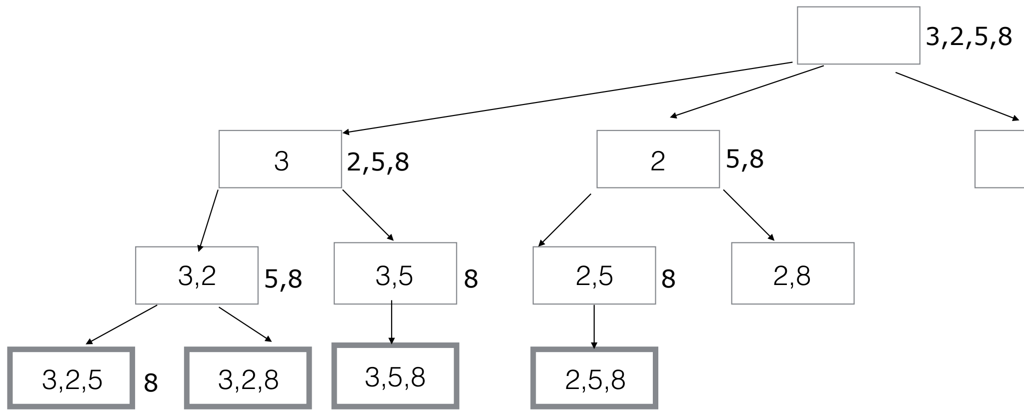
Backtracking



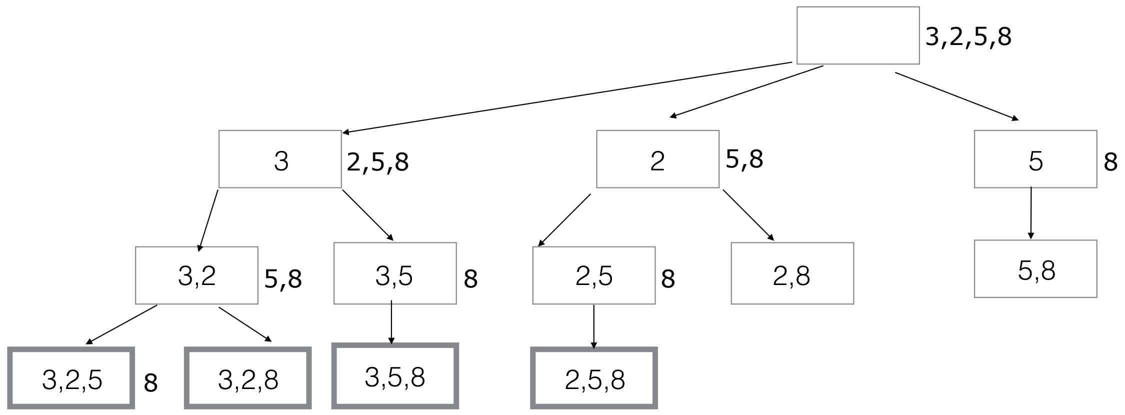
Backtracking



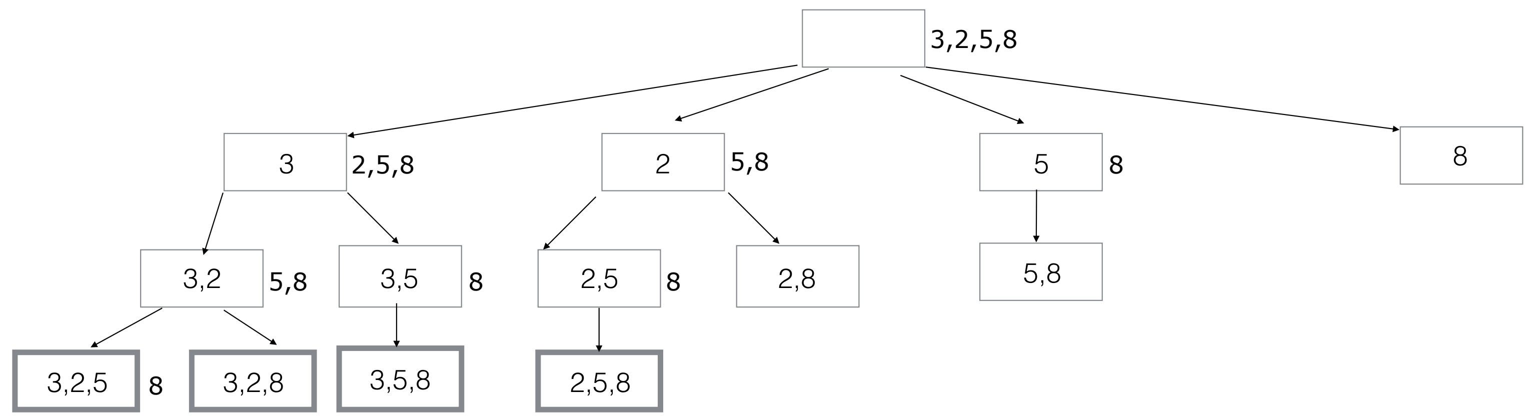
Backtracking



Backtracking



Backtracking



Java

```java
public static void combination(int[] input,int k,
HashSet<Integer> set,int start){

      if(set.size() == k){

          System.out.println(set);

          return;

      }
```

Java

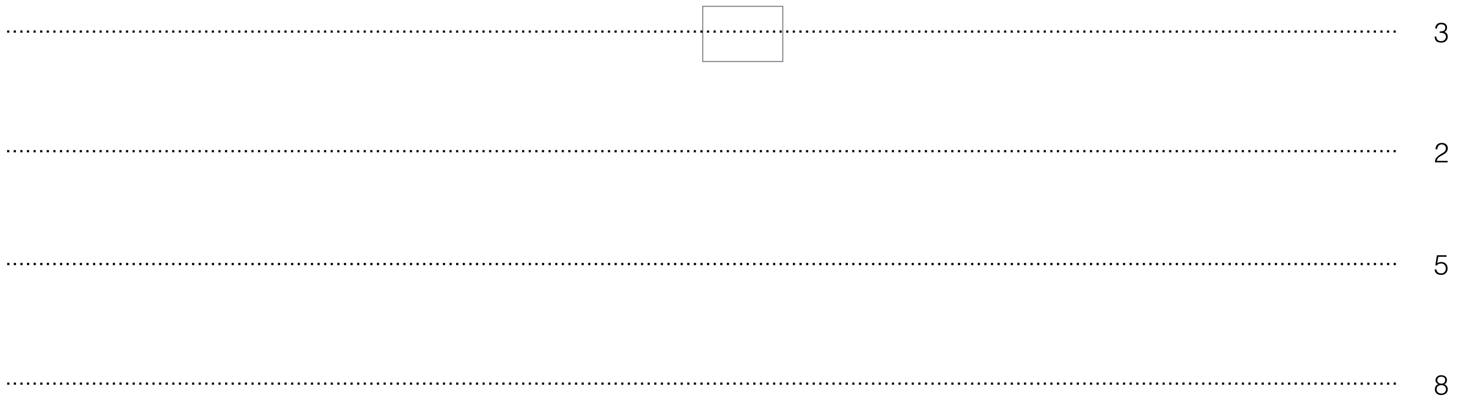
```java
      if(start == input.length){
           return;
       }
       for(int i=start;i<input.length;i++){
           set.add(input[i]);
           combination(input,set,i+1,k);
           set.remove(input[i]);
       }
}
```

Python

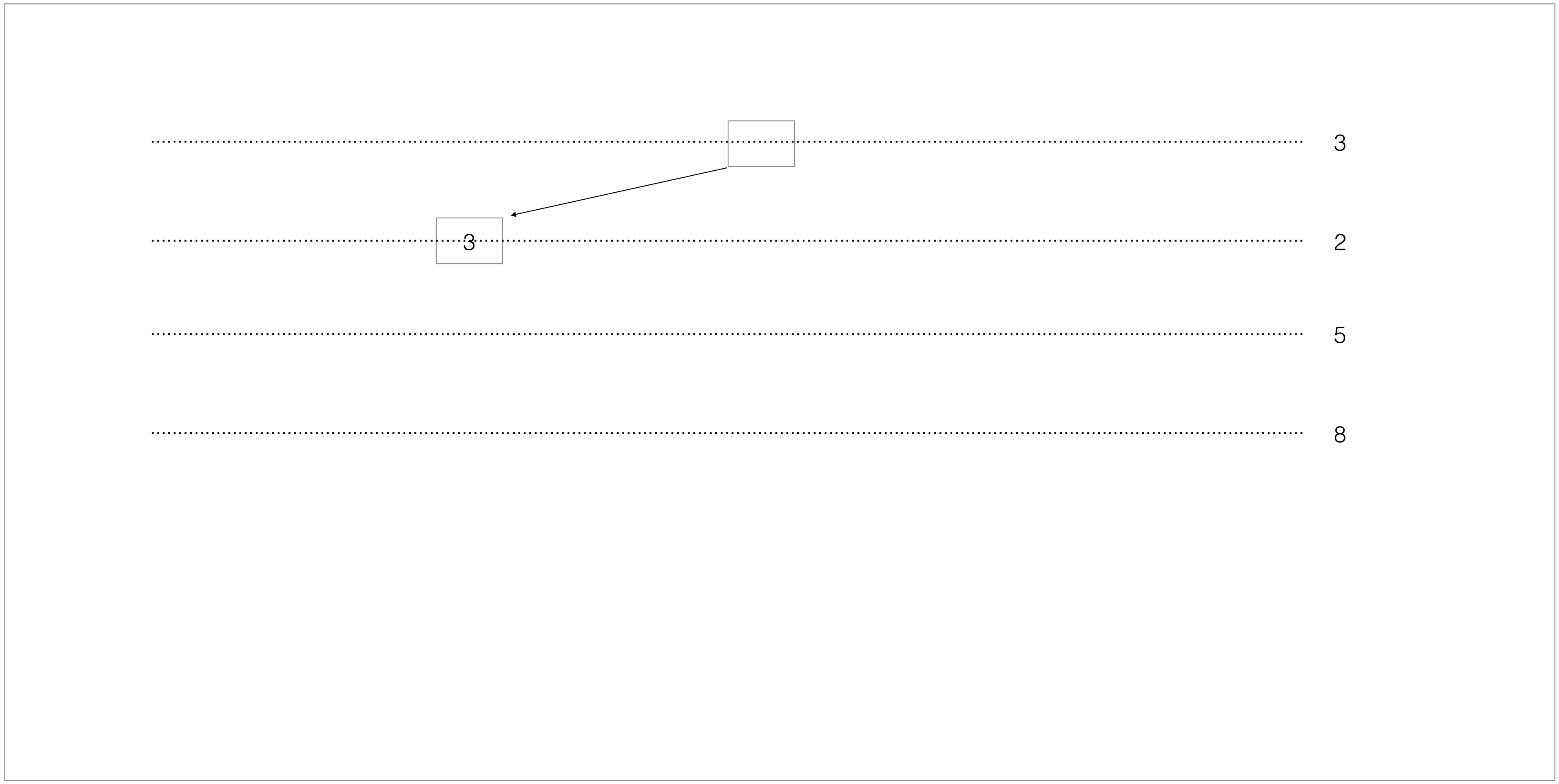
```python
def choose(input_list, k):
    combination(input_list, set(), 0, k)


def combination(input_list, comb, start, k):
    if len(comb) == k:
        print(comb)
        return
    if start == len(input_list):
        return
    for i in range(start, len(input_list)):
        comb.add(input_list[i])
        combination(input_list, comb, i + 1, k)
        comb.remove(input_list[i])
```

- Slightly different approach for creating subsets
- We go through one element at a time. We have two choices, either we include the integer in the set or not include it.

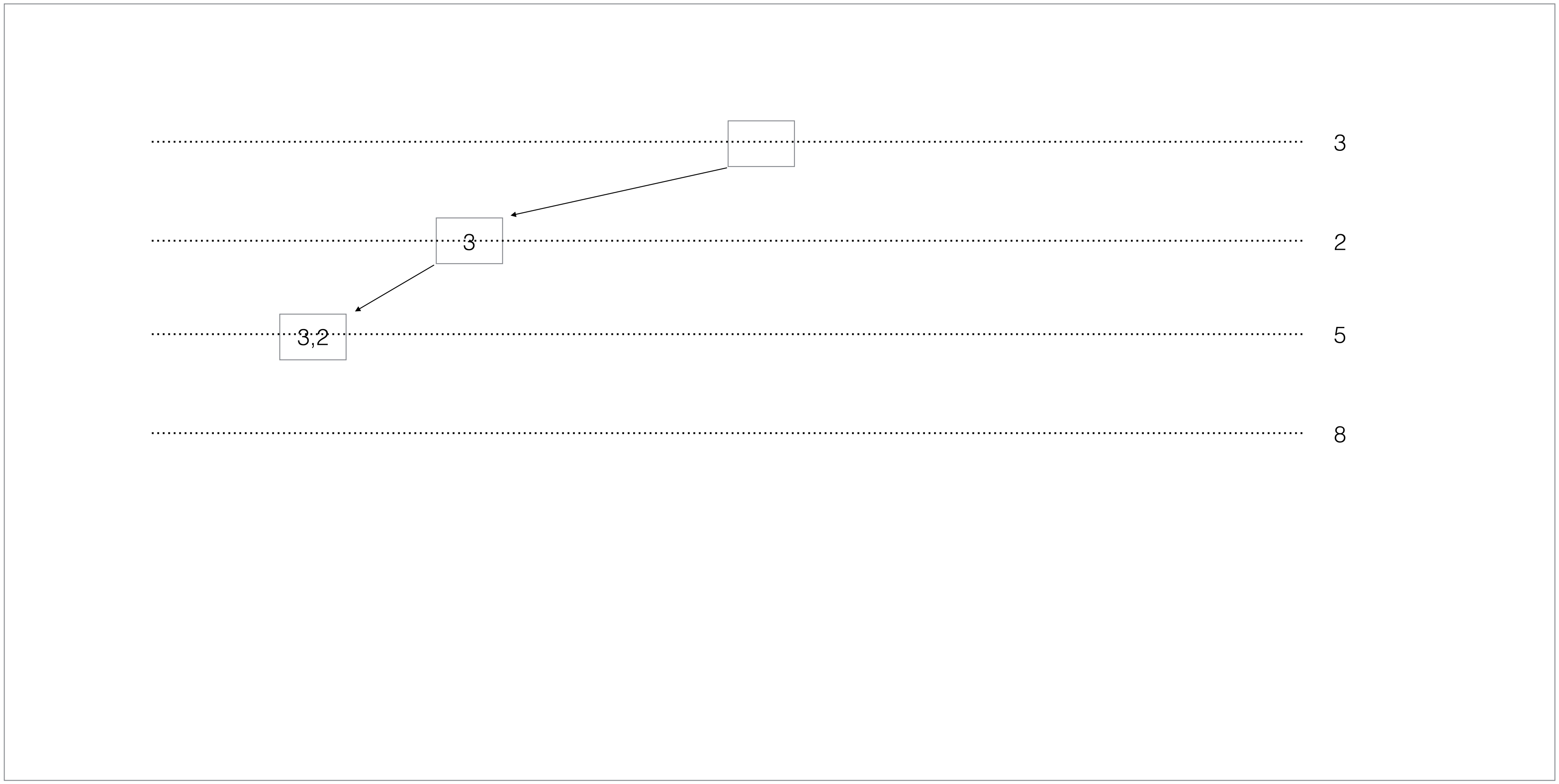
Backtracking



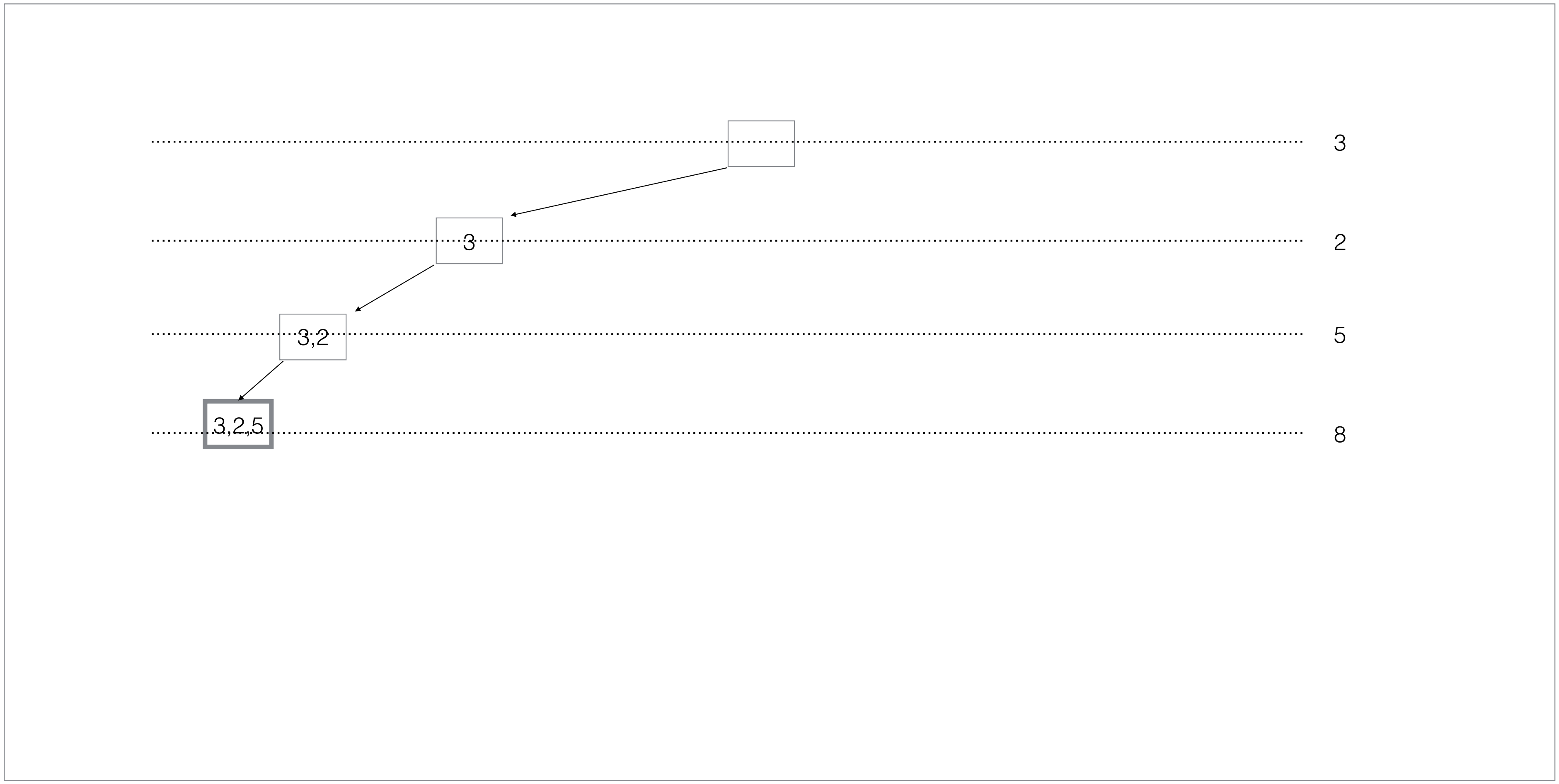
Backtracking



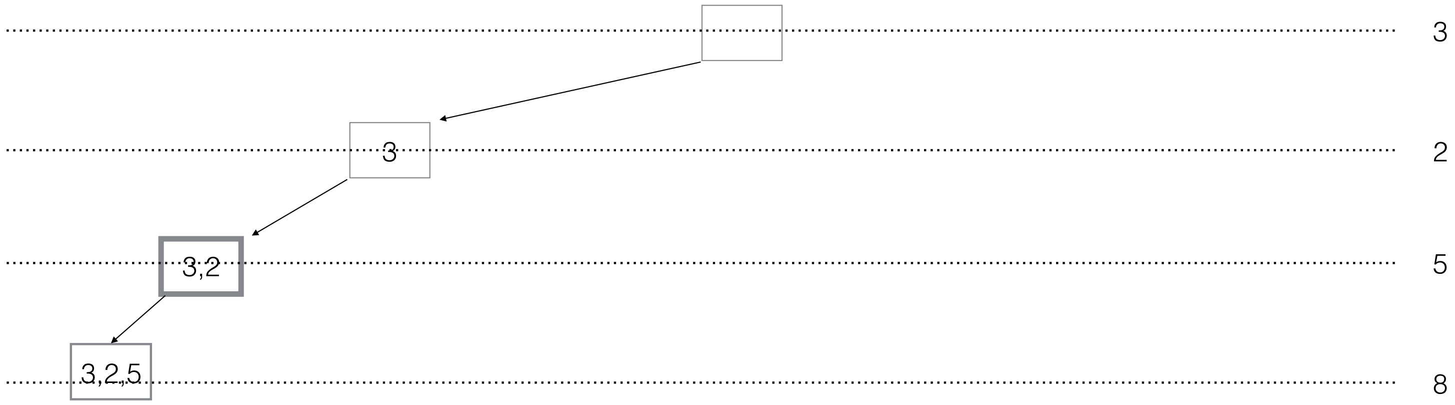
Backtracking



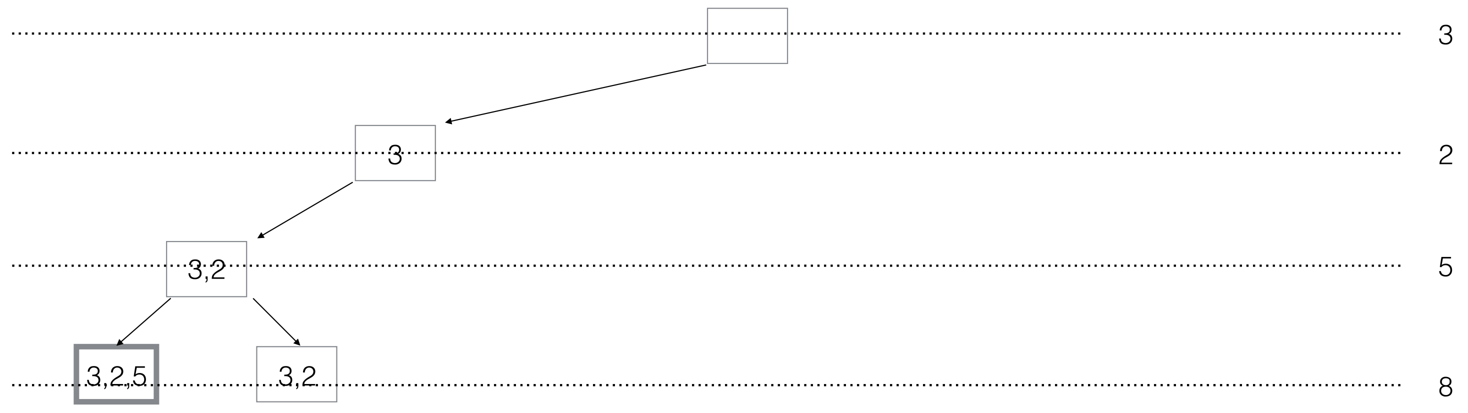
Backtracking



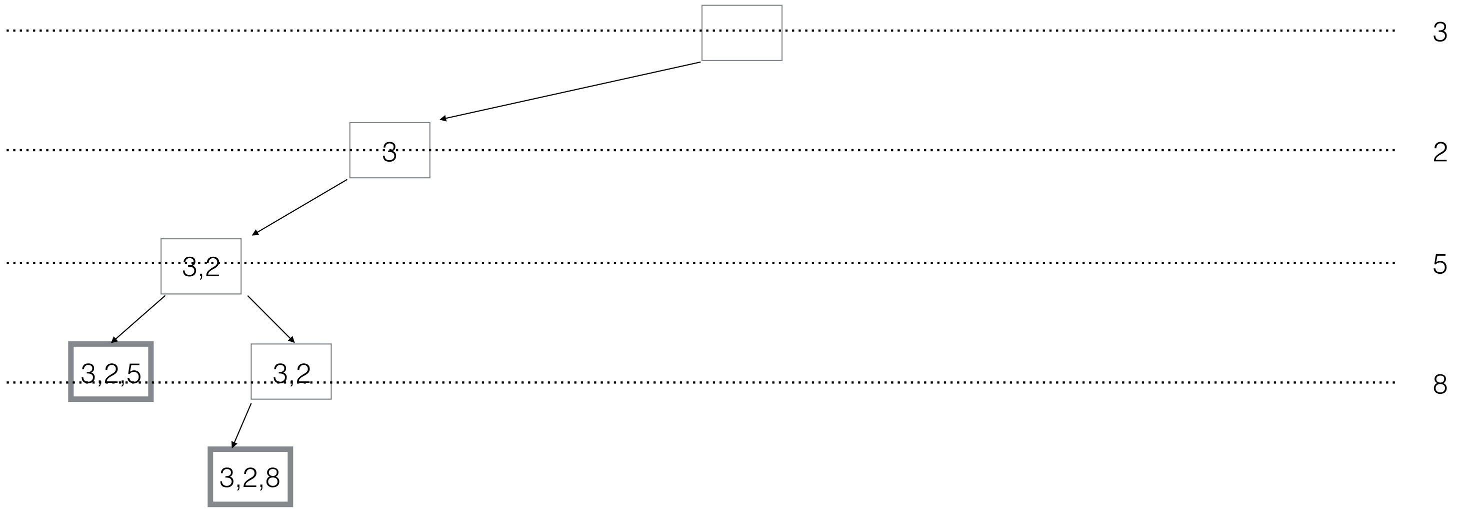
Backtracking



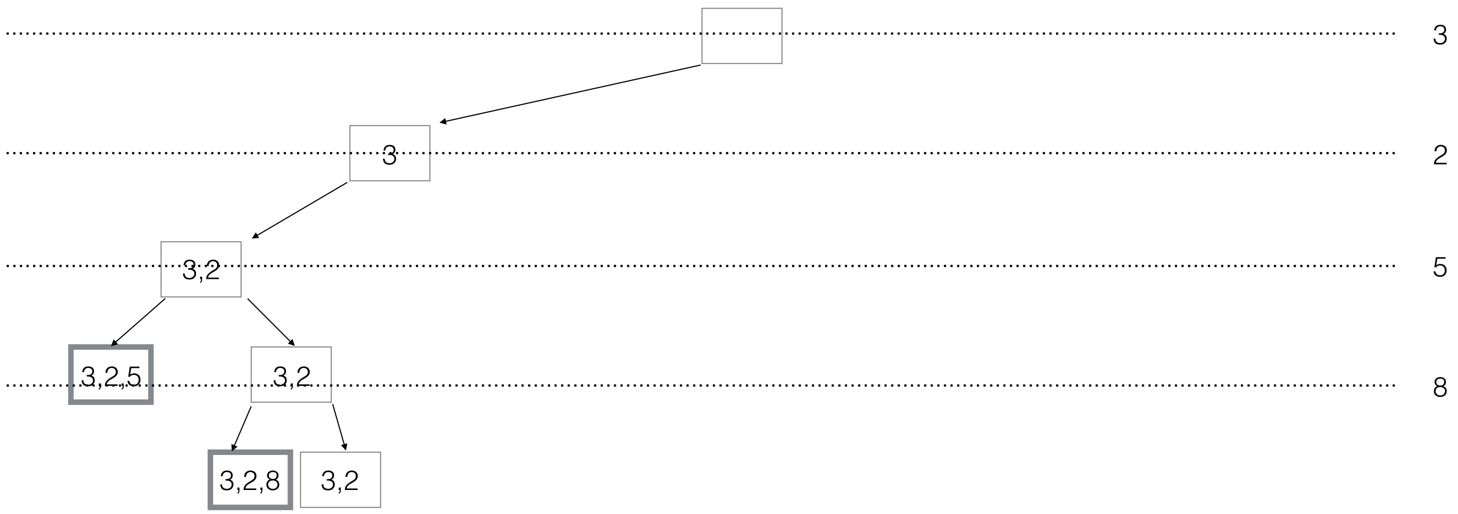
Backtracking



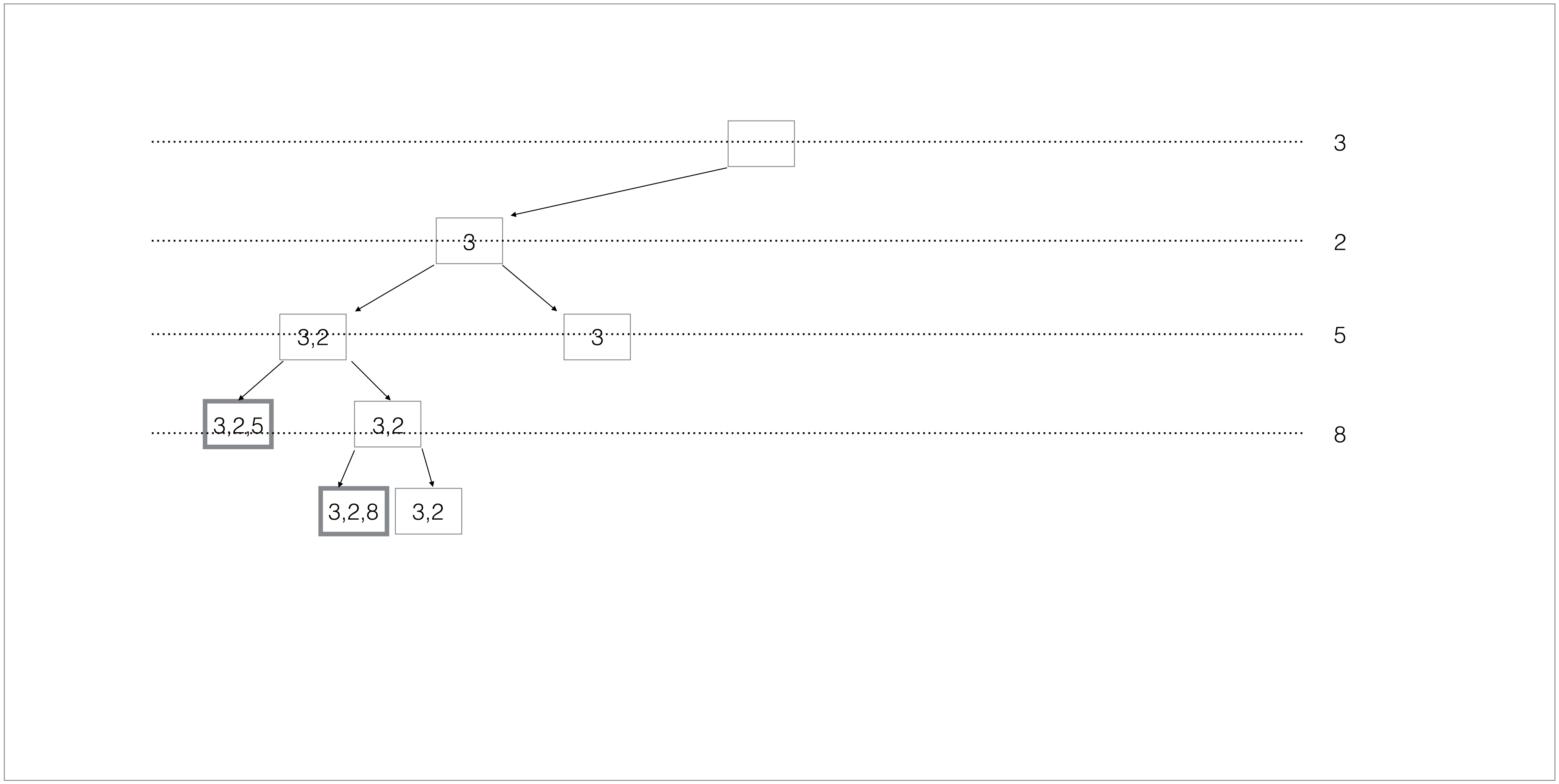
Backtracking



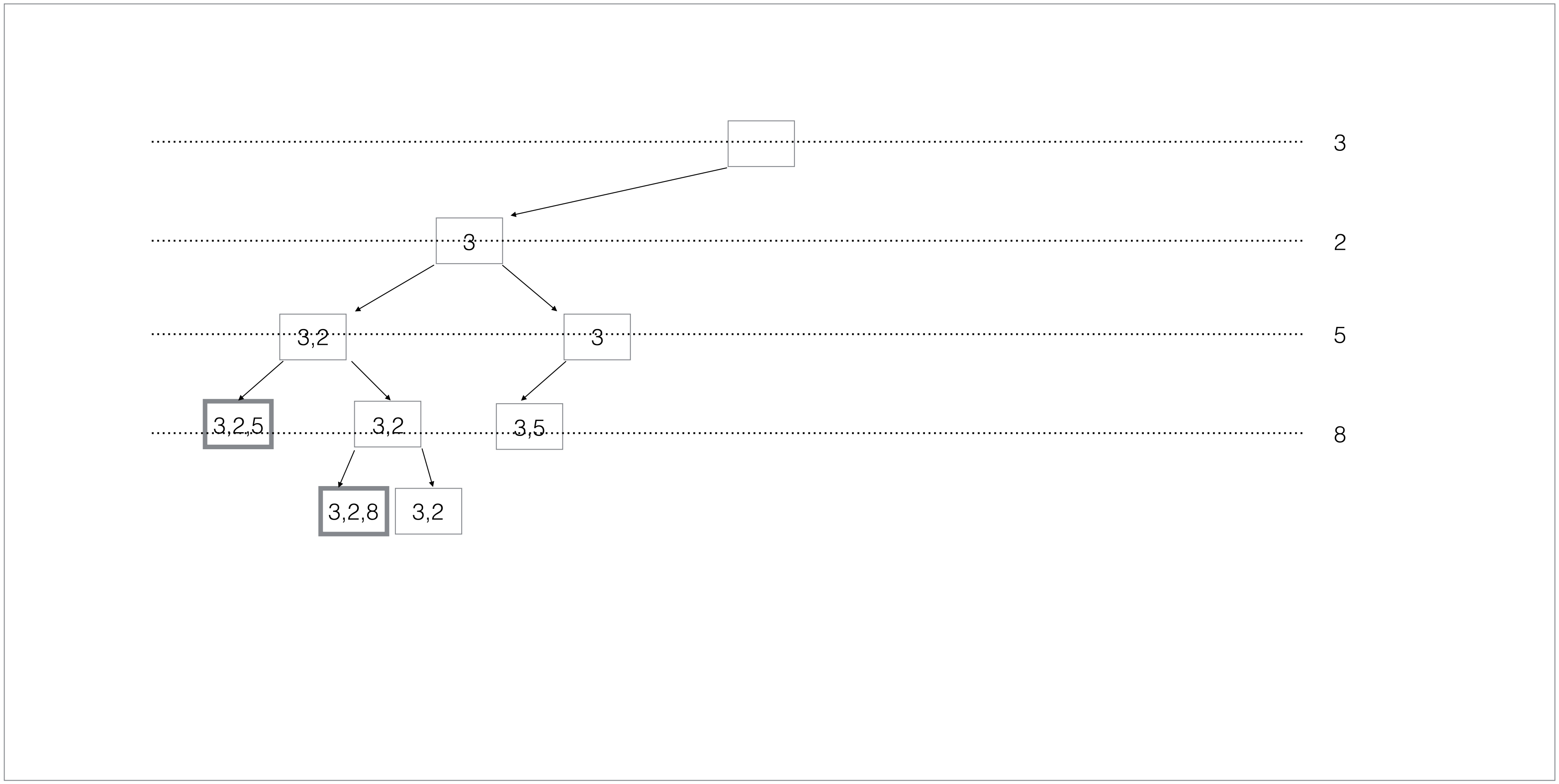
Backtracking



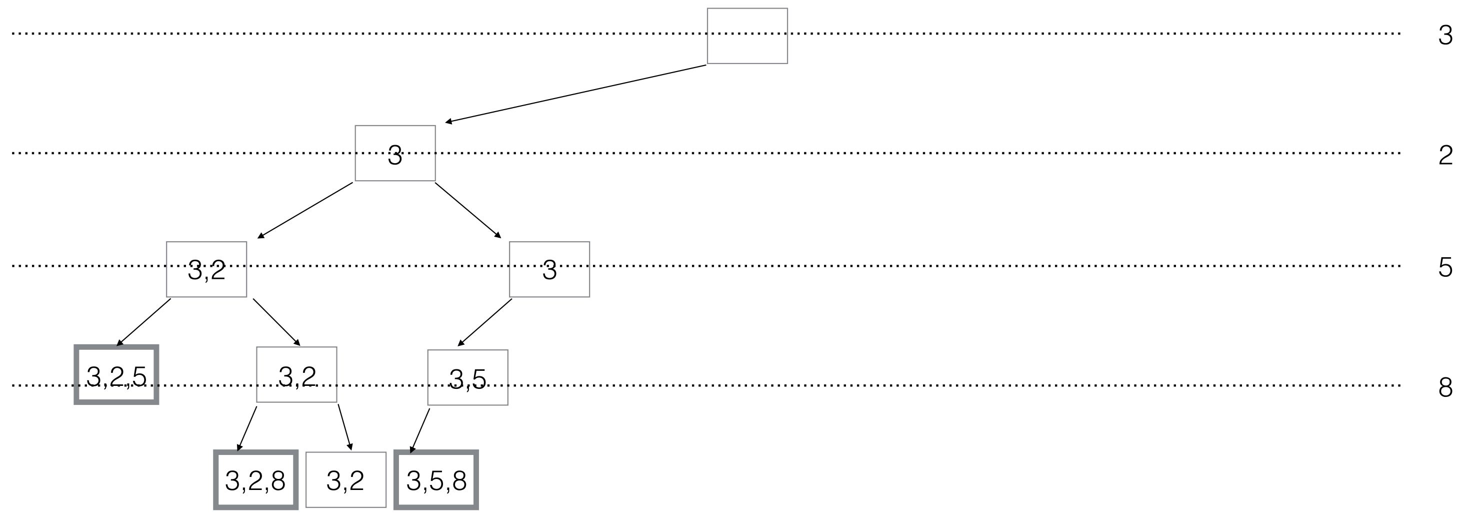
Backtracking



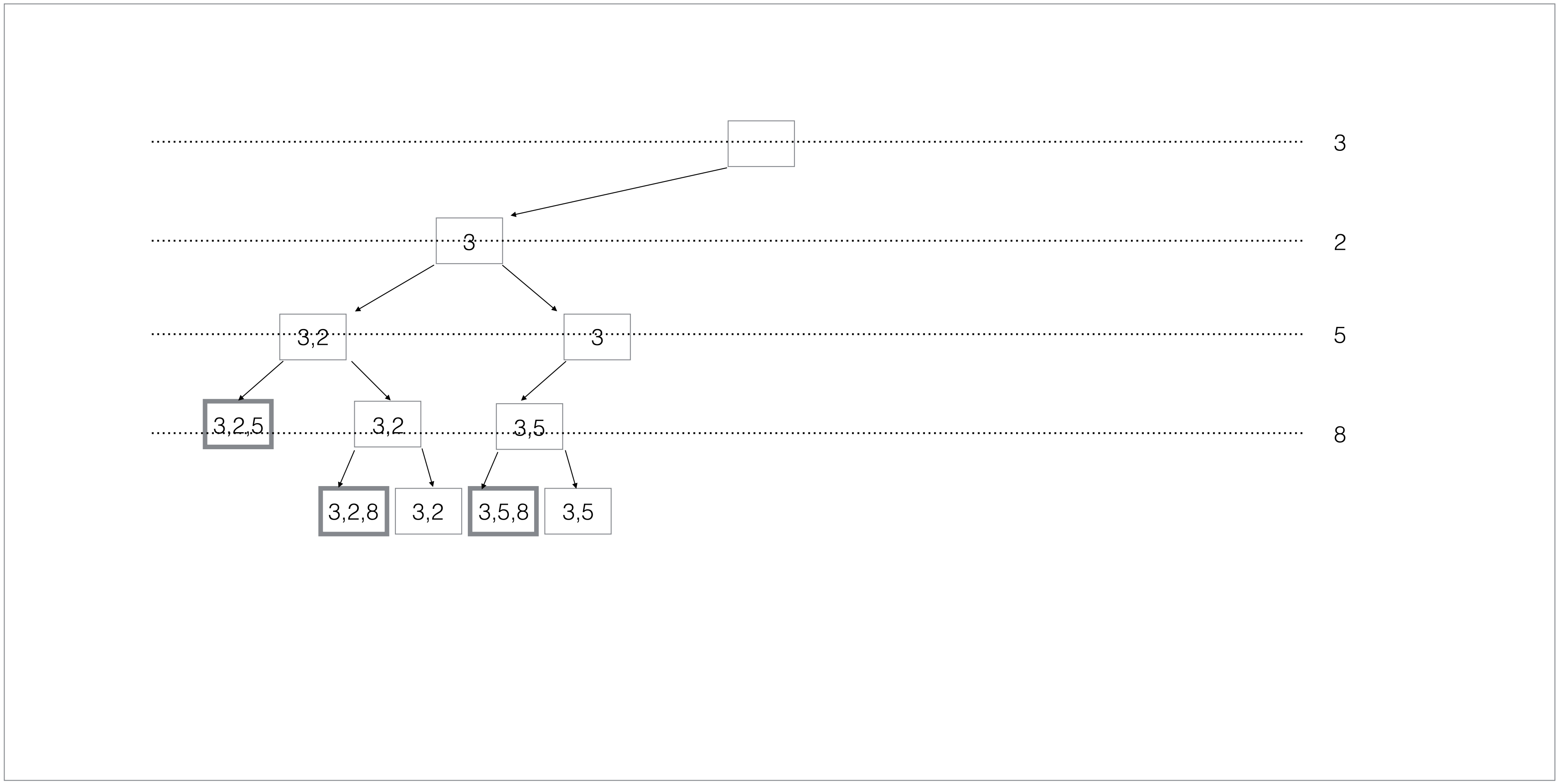
Backtracking



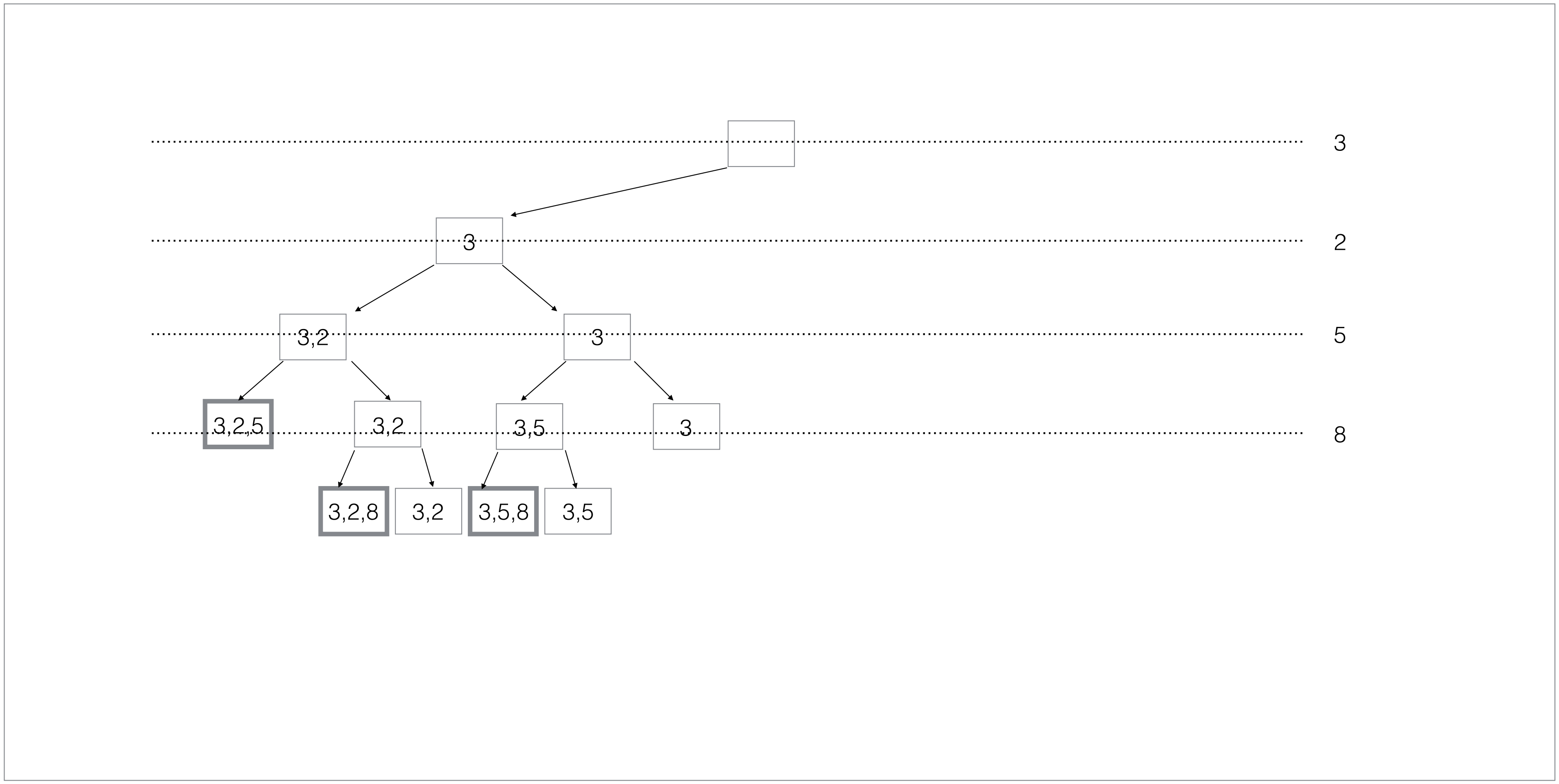
Backtracking



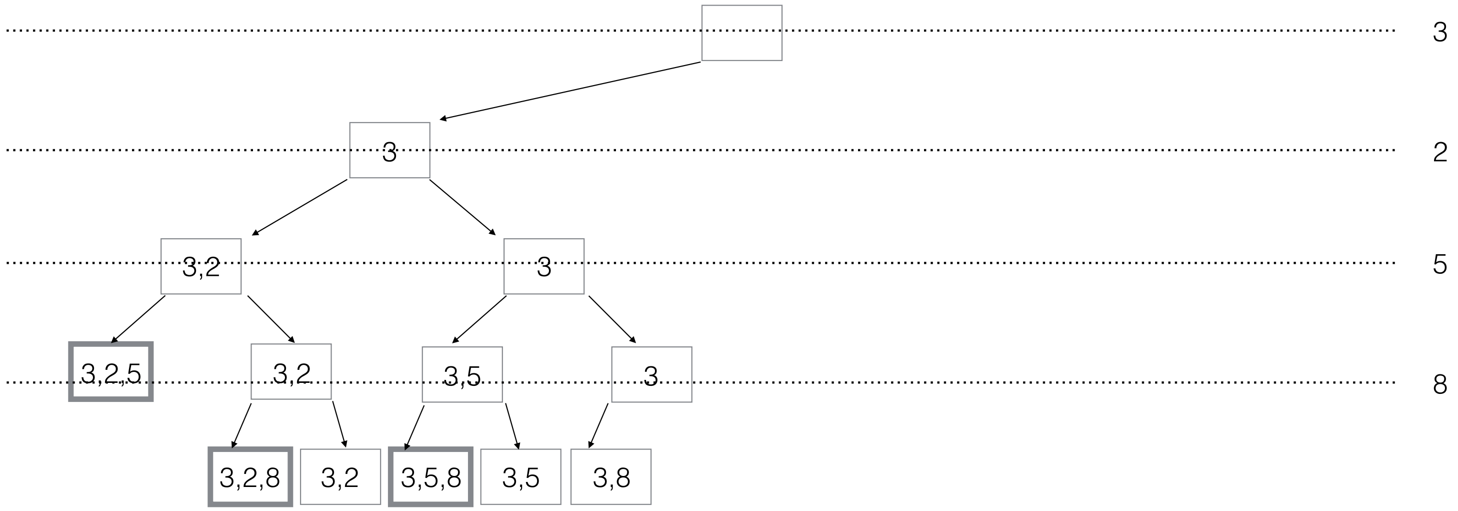
Backtracking



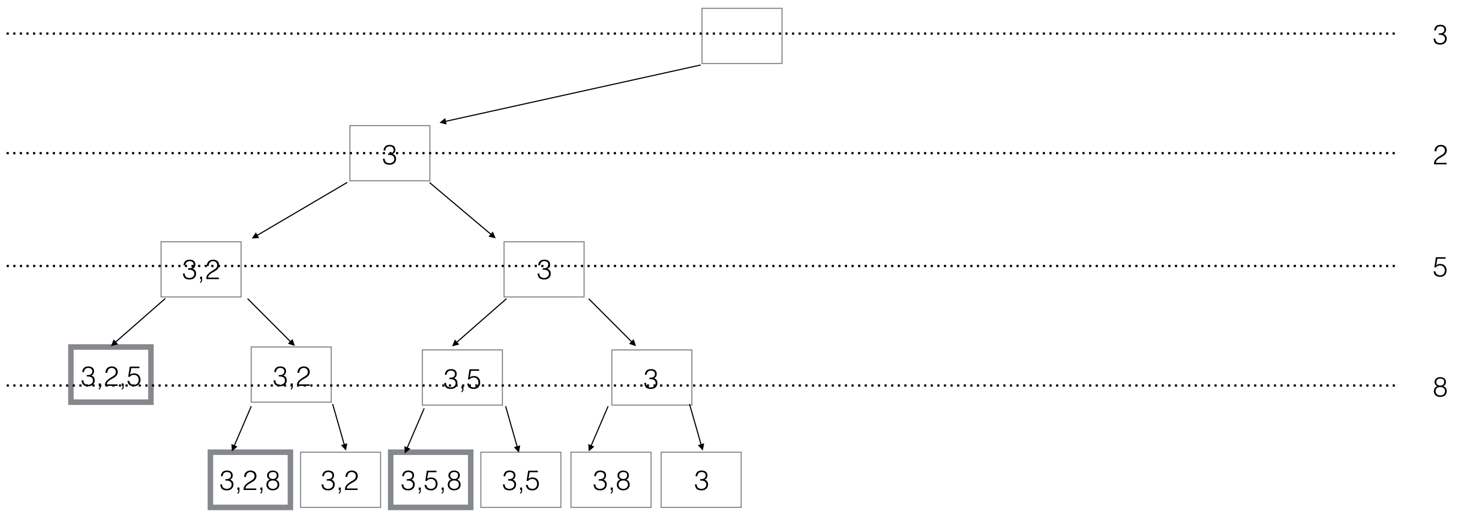
Backtracking



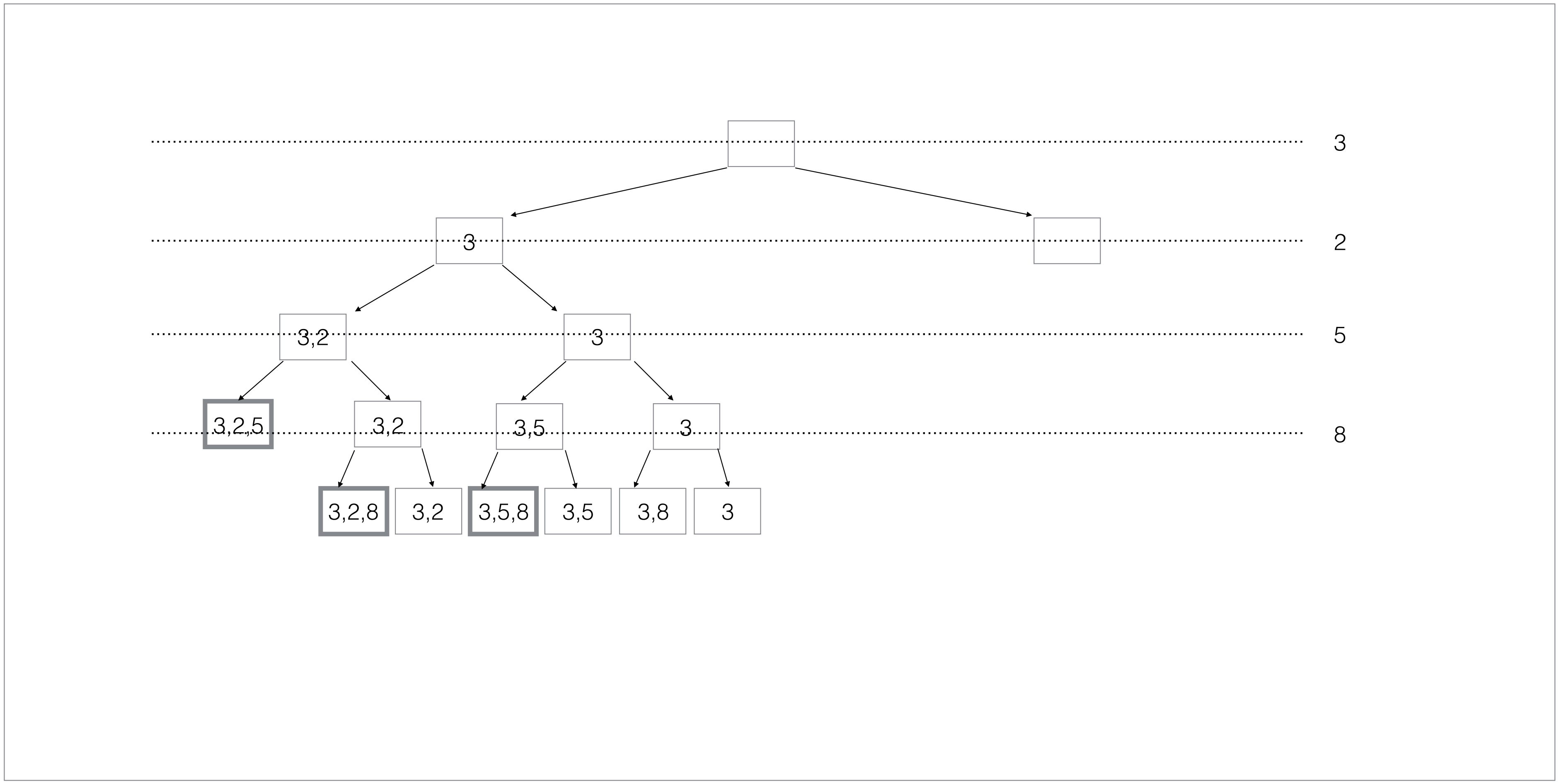
Backtracking



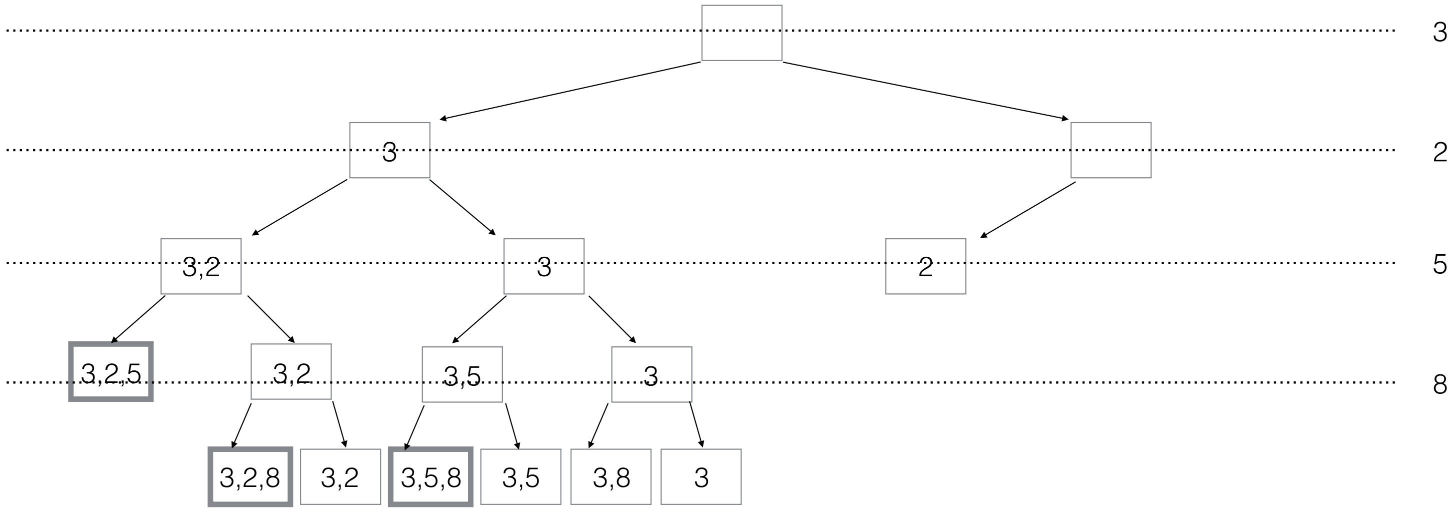
Backtracking



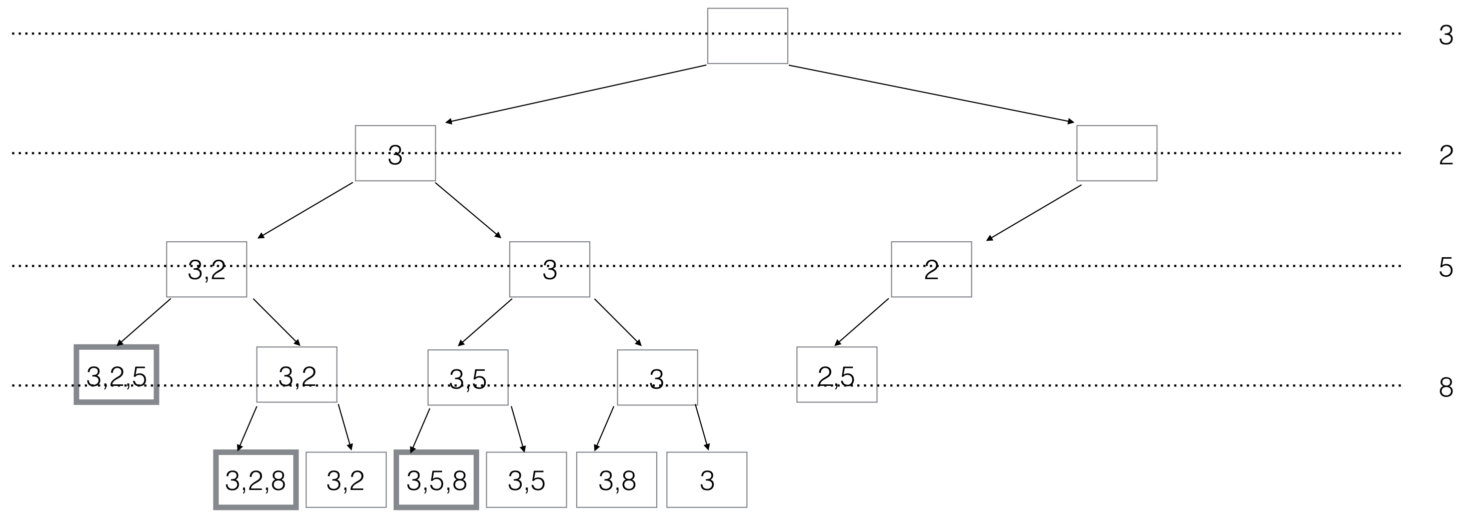
Backtracking



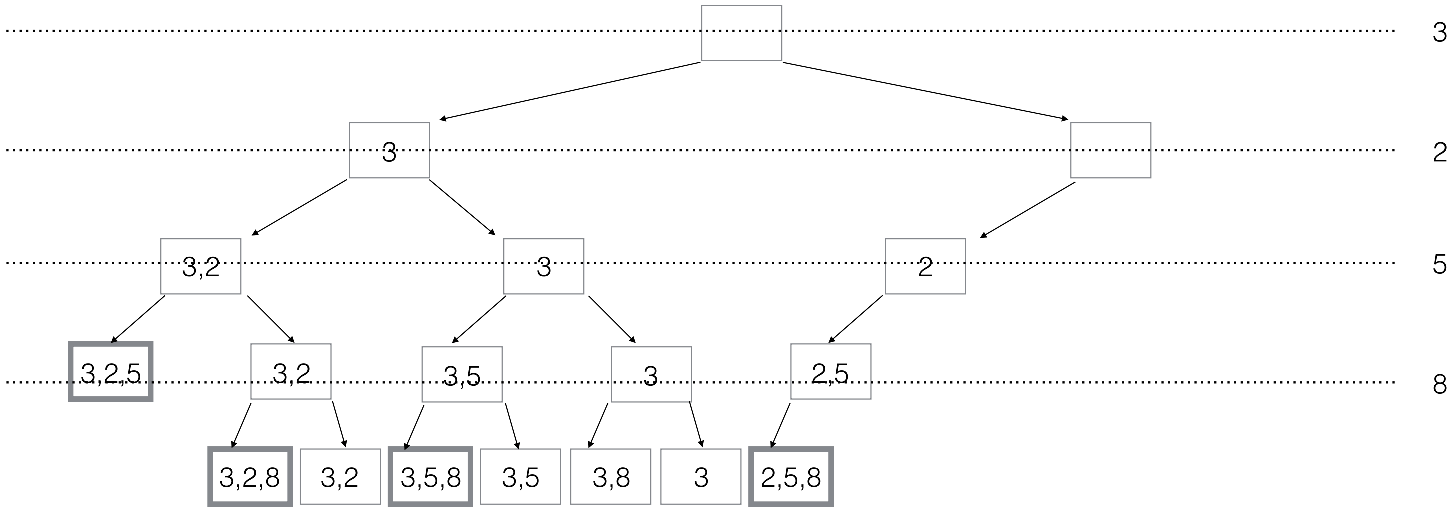
Backtracking



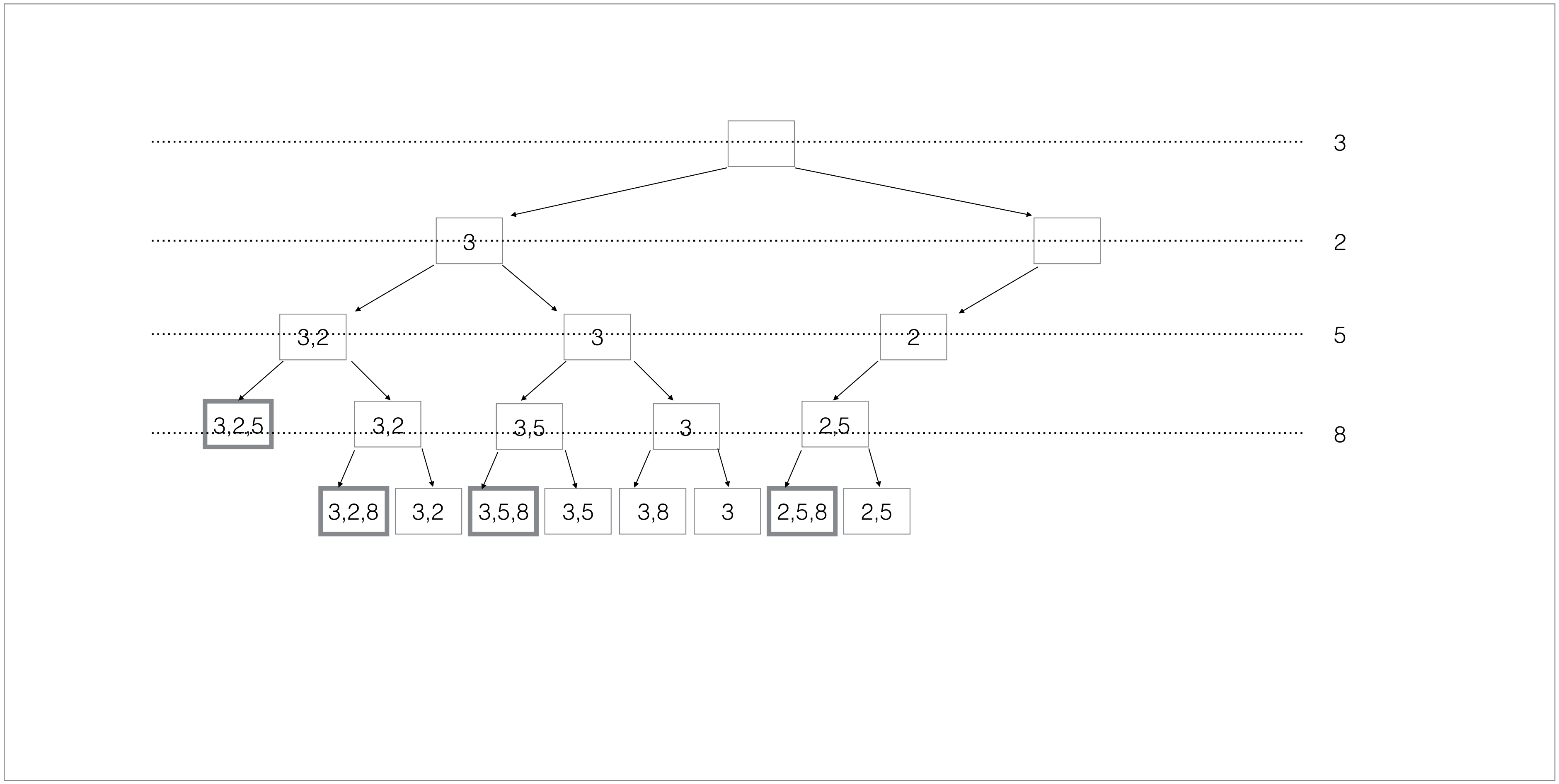
Backtracking



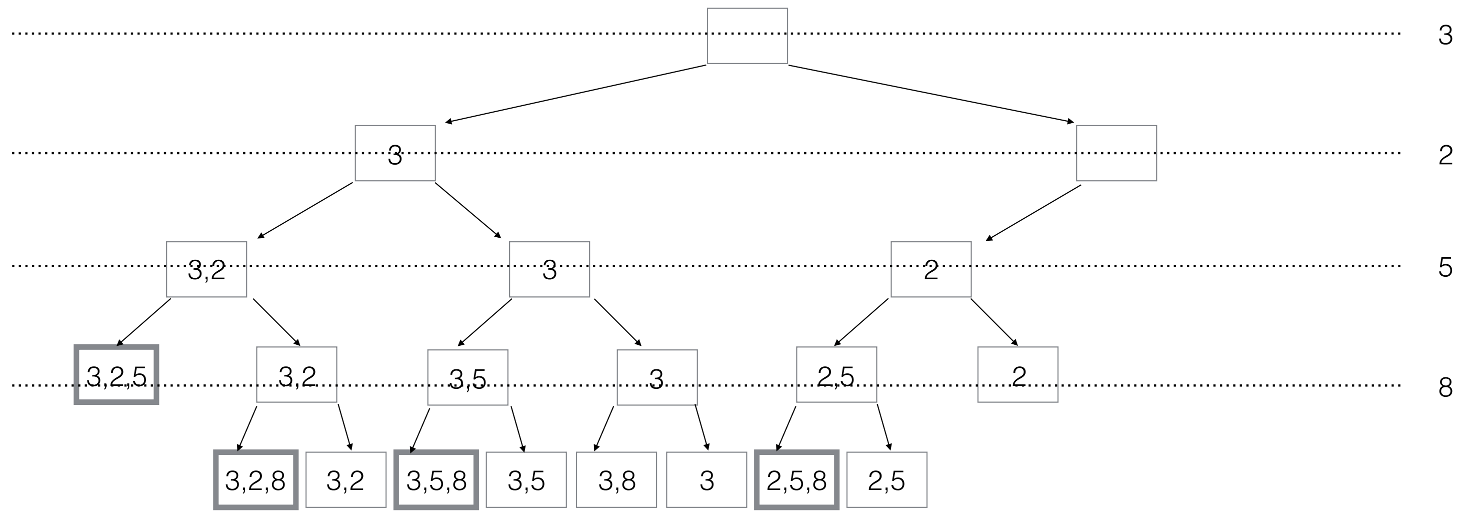
Backtracking



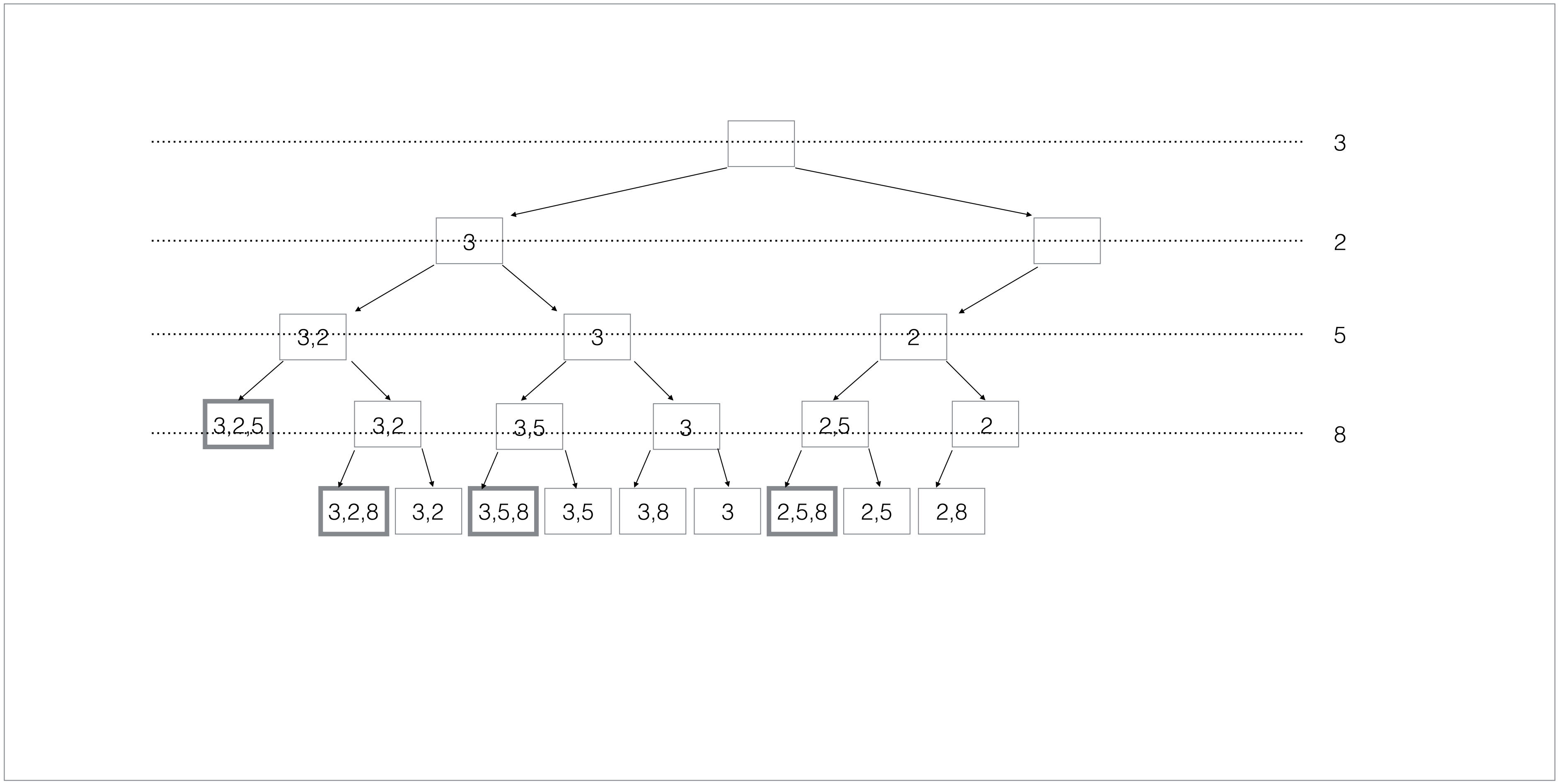
Backtracking



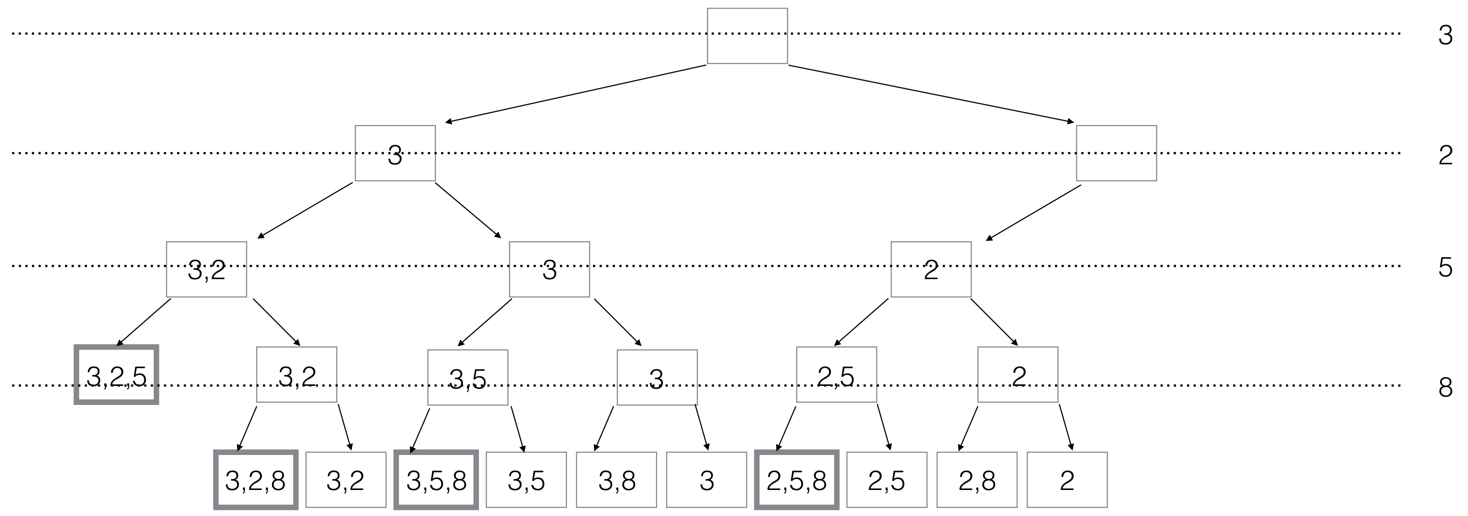
Backtracking



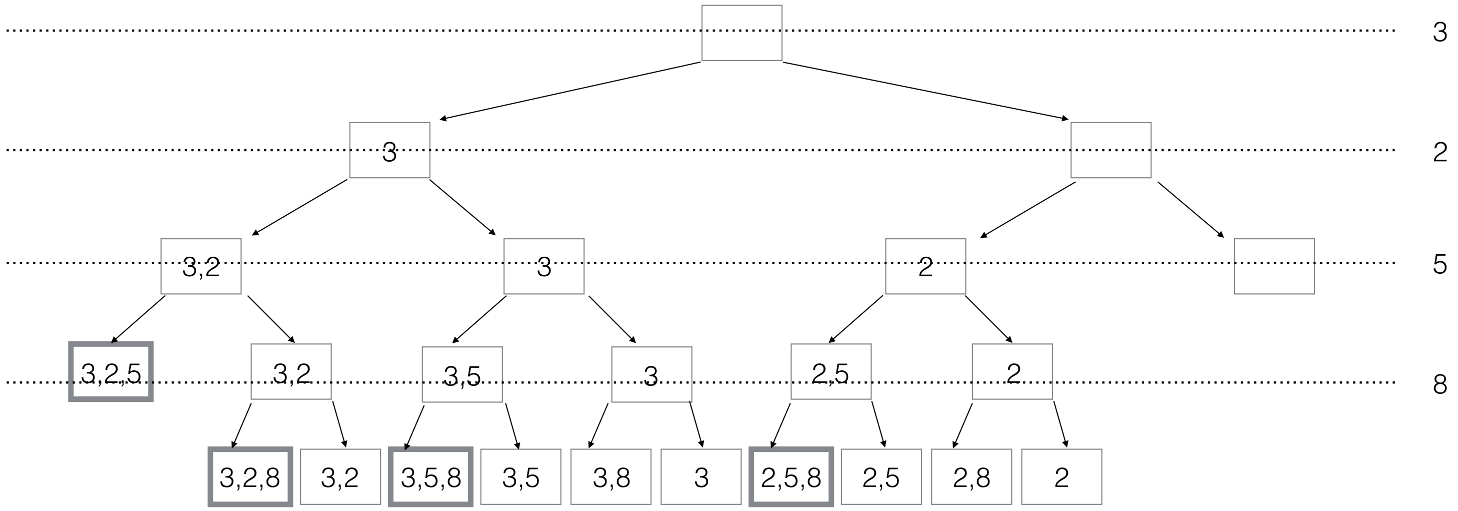
Backtracking



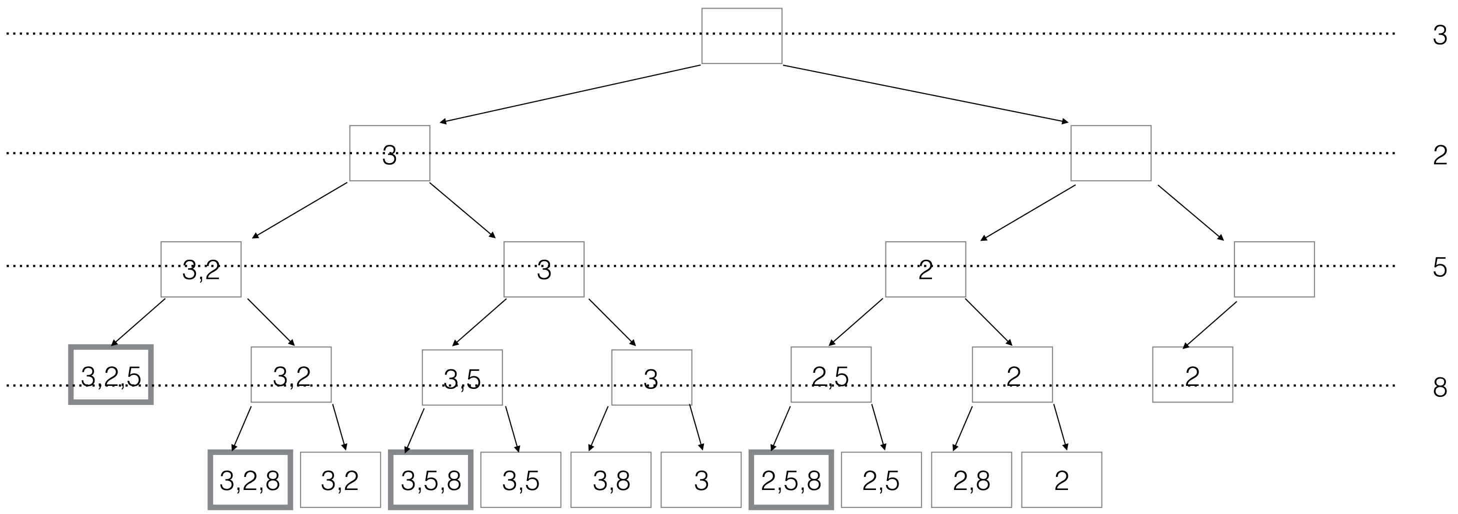
Backtracking



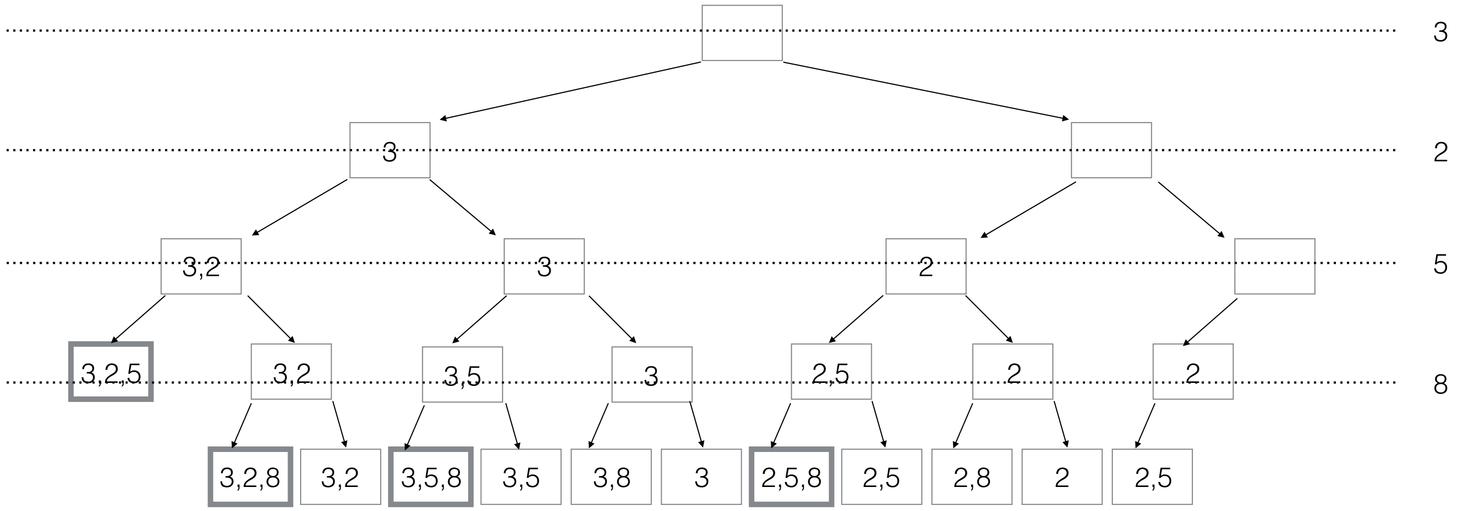
Backtracking



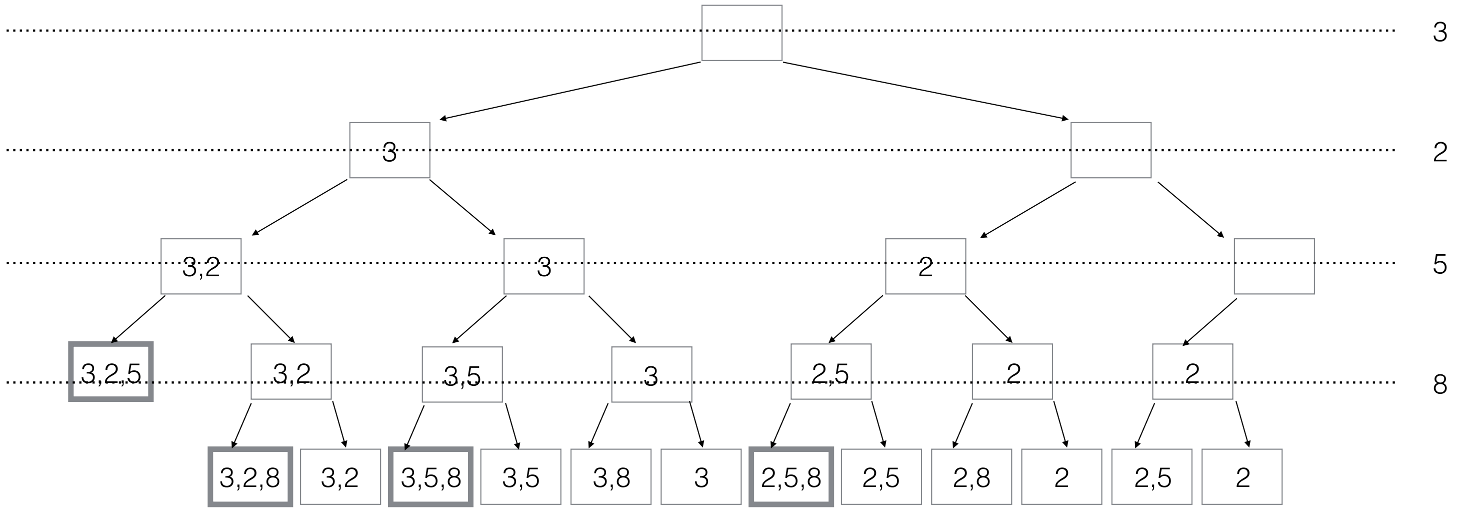
Backtracking



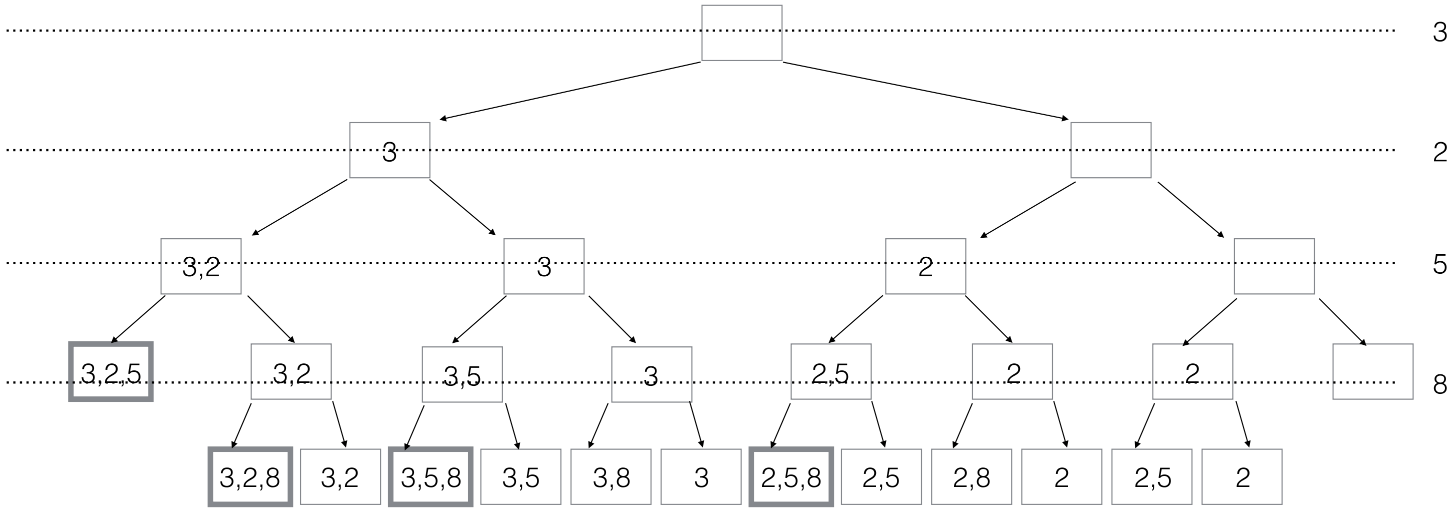
Backtracking



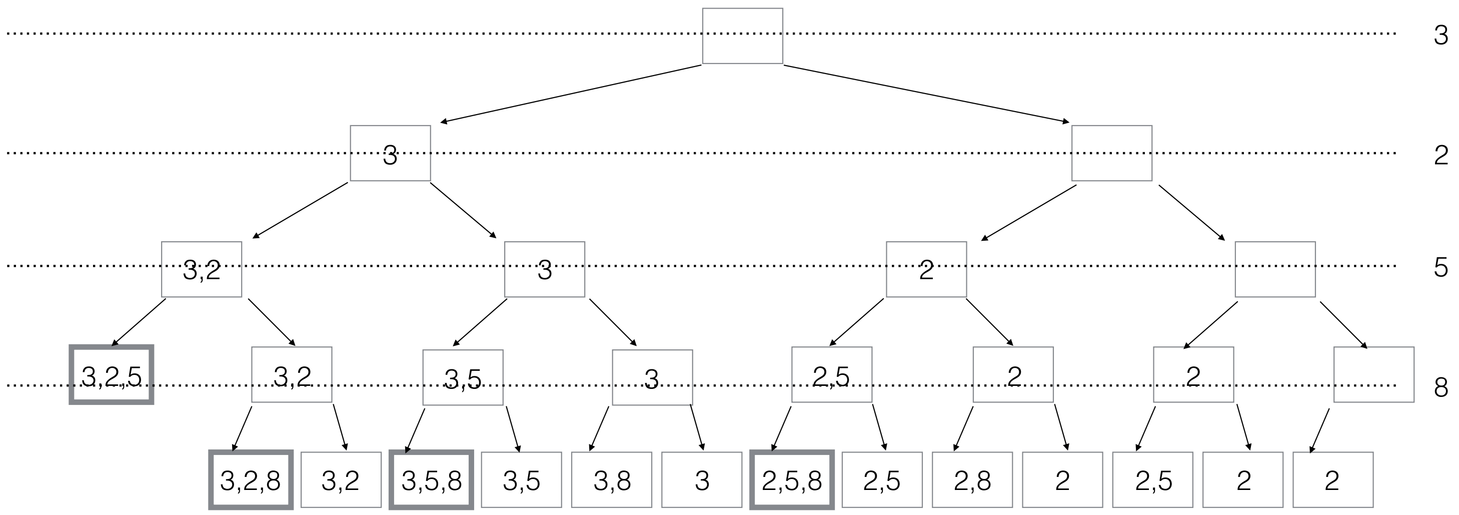
Backtracking



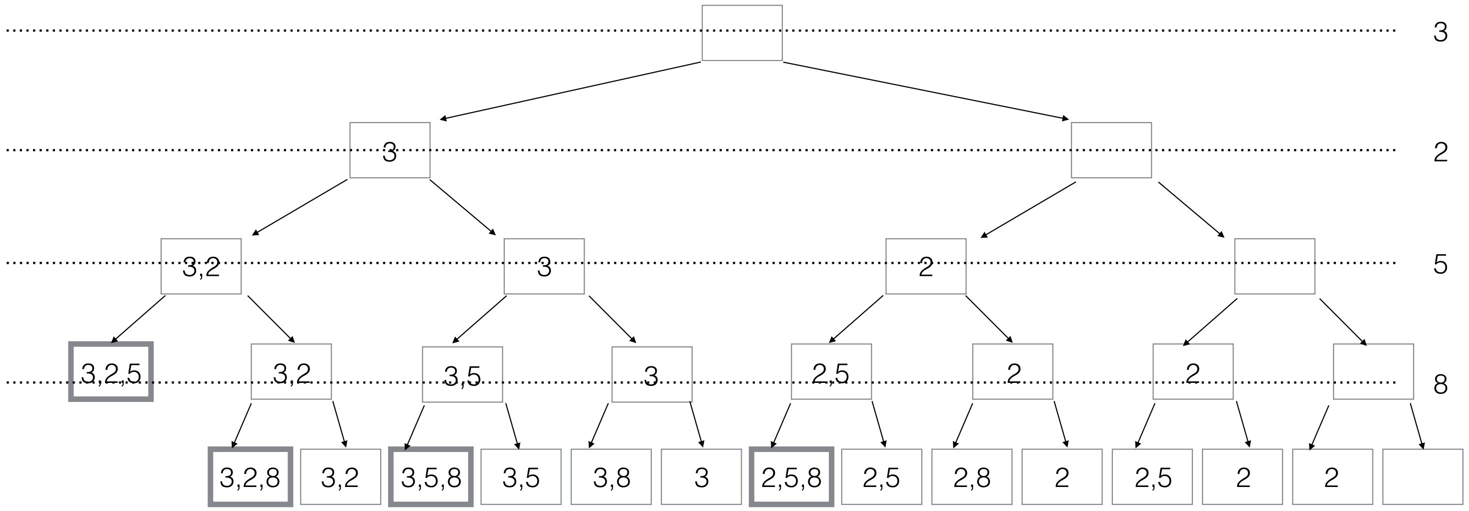
Backtracking



Backtracking



Backtracking



Java

```java
public static void combinationAlternative(int[]
input,HashSet<Integer> partial,int i,int k){
    if(partial.size() == k){
        System.out.println(Arrays.toString(partial.toArray()));
        return;
    }
    if(i == input.length){
        return;
    }
    partial.add(input[i]);
    combinationAlternative(input,partial,i+1,k);
    partial.remove(input[i]);
    combinationAlternative(input,partial,i+1,k);
}
```

Python

```python
def combination_alternative(input_list, comb, i, k):
    if len(comb) == k:
        print(comb)
        return
    if i == len(input_list):
        return
    comb.add(input_list[i])
    combination_alternative(input_list, comb, i + 1, k)
    comb.remove(input_list[i])
    combination_alternative(input_list, comb, i + 1, k)
```

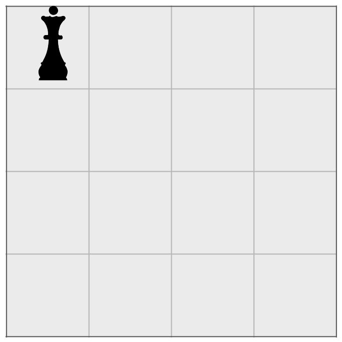
3. N queens problem

You are given a chess board of size NXN, where N is atmost 8.

You have to place N quees in this board such that no two queens attack. A Queen can move horizontally, vertically and Diagonally in any direction. Write a function given which takes N as argument and prints out the positions of N queens.

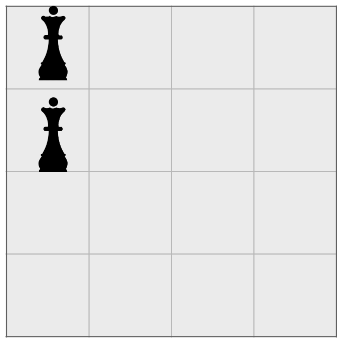
Backtracking

N=4



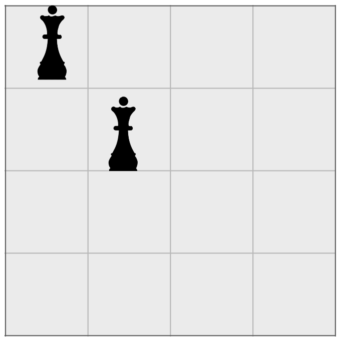
Backtracking

N=4



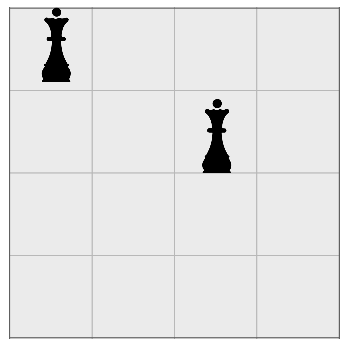
Backtracking

N=4



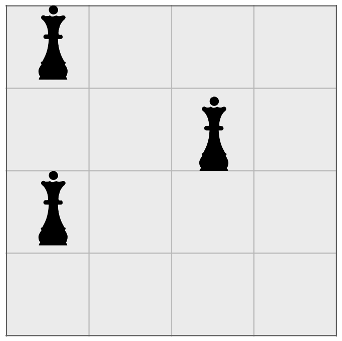
Backtracking

N=4



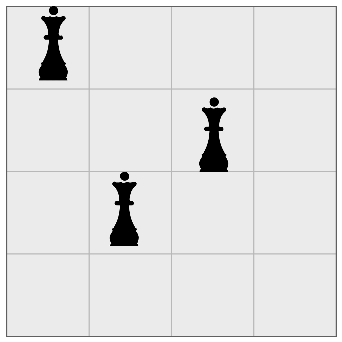
Backtracking

N=4



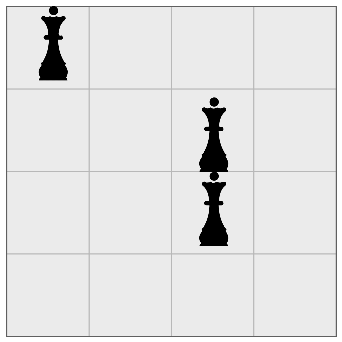
Backtracking

N=4



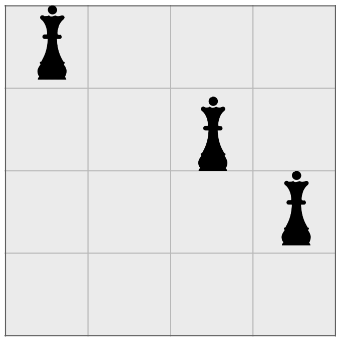
Backtracking

N=4



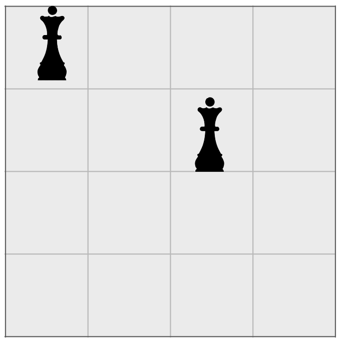
Backtracking

N=4



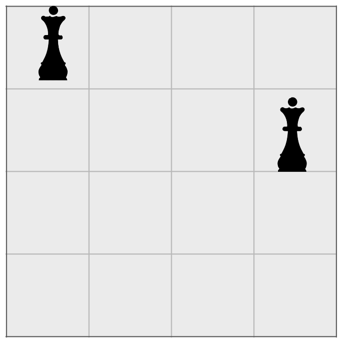
Backtracking

N=4



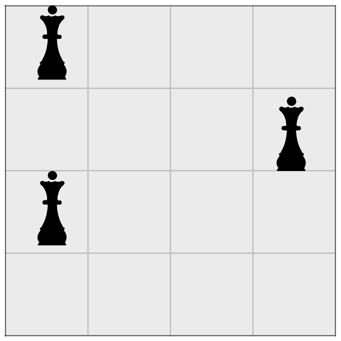
Backtracking

N=4



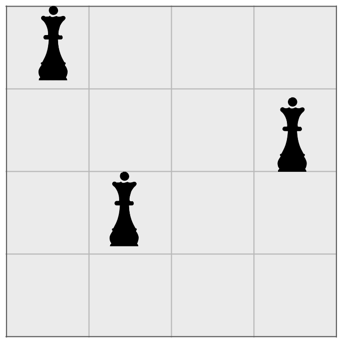
Backtracking

N=4



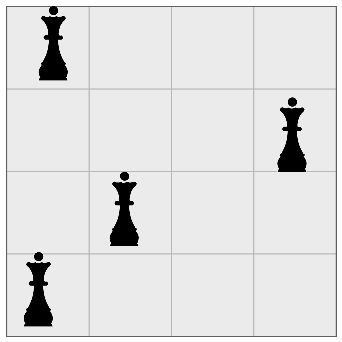
Backtracking

N=4



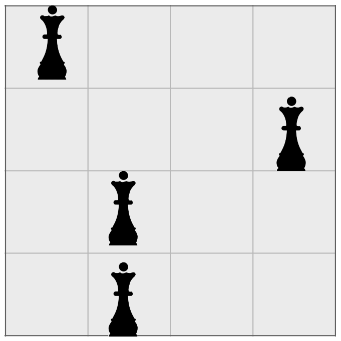
Backtracking

N=4



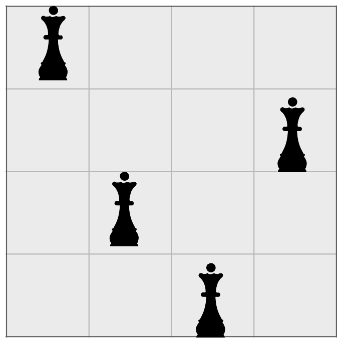
Backtracking

N=4



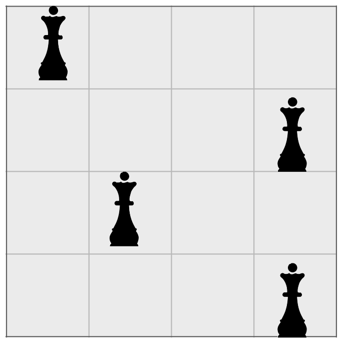
Backtracking

N=4



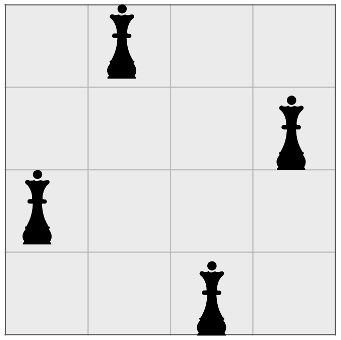
Backtracking

N=4



Backtracking

N=4



Java

```java
public static boolean helper(int[] board,int i){
    if(i == board.length){
        for(int row : board){
            for(int c=0;c<board.length;c++){
                if(c == row){
                    System.out.print(" O ");
                }else{
                    System.out.print(" X ");
                }
            }
            System.out.println("");
        }
        return true;
    }
```

Java

```java
    for(int c=0;c<board.length;c++){
        boolean flag = false;
        for(int r=0;r<i;r++){
            if(board[r] == c || Math.abs(board[r]-c) == (i-r)){
                flag = true;
                break;
            }
        }
        if(flag){
            continue;
        }
        board[i]=c;
        if(helper(board,i+1)){
            return true;
        }
    }
    return false;
}
```

Python

```python
def helper(board, i):
    n = len(board)
    if i == n:
        for c in board:
            for r in range(0, n):
                if r == c:
                    print("O", end=" ")
                else:
                    print("X", end=" ")
            print("")

        return True
```

Python

```
    for c in range(0, n):
        flag = False
        for rc in range(0, i):
            if c == board[rc] or abs(board[rc] - c) == i - rc:
                flag = True
                break
        if flag:
            continue

        board[i] = c
        if helper(board, i + 1):
            return True

    return False
```

4. Given a list of numbers , and a target number. Print all the unique combinations in candidates where the candidate numbers sum to target.

Example:

Input = [10,1,2,7,6,1,5] , target=8

Output:

[1, 1, 6]

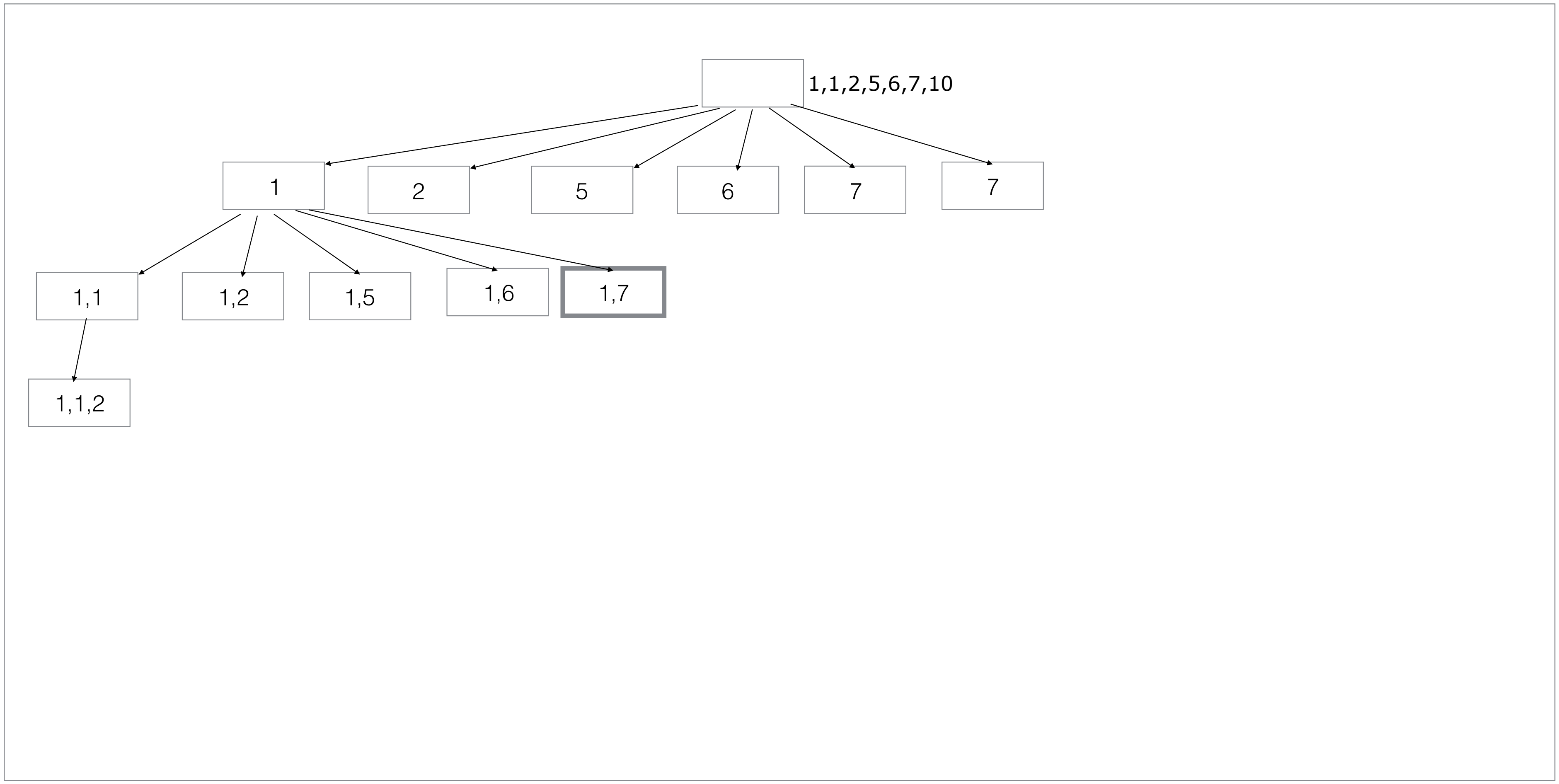
[1, 2, 5]

[1, 7]

[2, 6]

[1,2,5] = [2,1,5] = [5,1,2] , we should pick only one to avoid duplicates.

Target sum



Java

```java
public static void combinationSum(int[] nums, int target) {
    Arrays.sort(nums);
    combinationSum(nums, target, 0, new ArrayList<>(), 0);
}

public static void combinationSum(int[] nums, int target, int sum, List<Integer>
partial, int start) {
    if (sum == target) {
        System.out.println(Arrays.toString(partial.toArray()));
        return;
    }

    for (int i = start; i < nums.length; i++) {
        int c = nums[i];
        if (sum + c > target || i > start && nums[i] == nums[i - 1]) {
            continue;
        }
        partial.add(c);
        combinationSum(nums, target, sum + c, partial, i + 1);
        partial.remove(partial.size() - 1);
    }
}
```

Python

```python
def choose(input_list, target):
    input_list=sorted(input_list)
    combination(input_list,target, [], 0)


def combination(input_list,target, comb, start):
    if target == 0:
        print(comb)
        return
    if start == len(input_list):
        return
    for i in range(start, len(input_list)):
        cand = input_list[i]
        if cand > target or (i>start and cand == input_list[i-1]):
            continue
        comb.append(cand)
        combination(input_list,target-cand, comb, i + 1)
        comb.pop()
```

Exercise

1.Given a string, write a function to print out all its anagrams.

Example:

Input = “god”

Output:

“god”

“gdo”

“dog”

“dgo”

“ogd”

“odg”

Java

```java
public static void anagrams(String input) {
    char[] inputArray = input.toCharArray();
    Arrays.sort(inputArray);
    anagrams(inputArray, new char[input.length()], new boolean[input.length()], 0);
}

public static void anagrams(char[] input, char[] anagram, boolean[] used, int
index) {
    if (index == input.length) {
        System.out.println(new String(anagram));
        return;
    }
    for (int i = 0; i < input.length; i++) {
        if (!used[i] && !(i > 0 && input[i] == input[i - 1] && !used[i - 1])) {
            used[i] = true;
            anagram[index] = input[i];
            anagrams(input, anagram, used, index + 1);
            used[i] = false;
        }
    }
}
```

Python

```python
def anagrams(input_string, anagram, used, index):
    if index == len(input_string):
        print(anagram)
        return
    for i in range(0, len(input_string)):
        if not used[i]:
            used[i] = True
            anagram[index] = input_string[i]
            anagrams(input_string, anagram, used, index + 1)
            used[i] = False
```

2. Given a string s and a dictionary containing a list of words, write a function to break the string completely into valid words. Print all such possible sentences. The same word in the dictionary may be reused multiple times.

For example

Input="catsanddog"

dictionary = ["cat", "cats", "and", "sand", "dog"]

Output:

“cat”, “sand”, “dog"

“cats”, “and”, dog”

Backtracking

Input = "catsanddog"
Dict = ["cat", "cats", "and", "sand", "dog"]

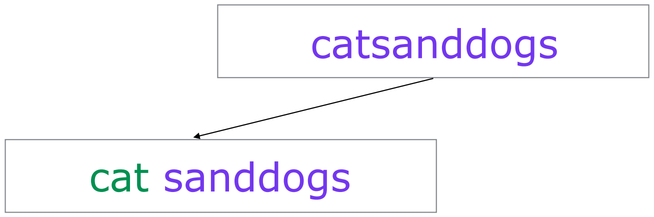
Backtracking

Input = "catsanddog"
Dict = ["cat", "cats", "and", "sand", "dog"]

catsanddogs

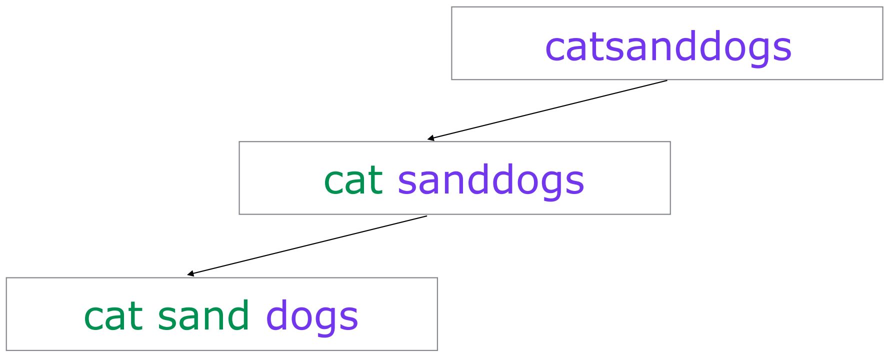
Backtracking

Input = "catsanddog"
Dict = ["cat", "cats", "and", "sand", "dog"]



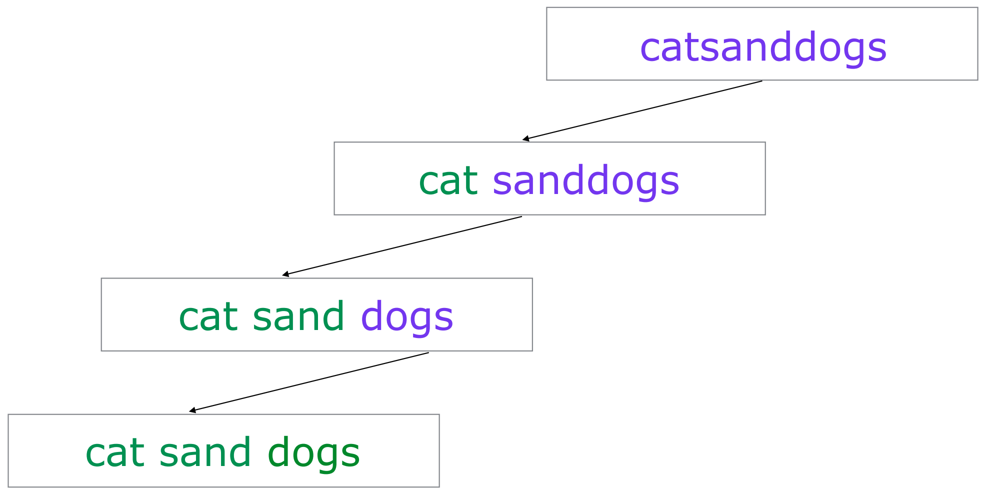
Backtracking

Input = "catsanddog"
Dict = ["cat", "cats", "and", "sand", "dog"]



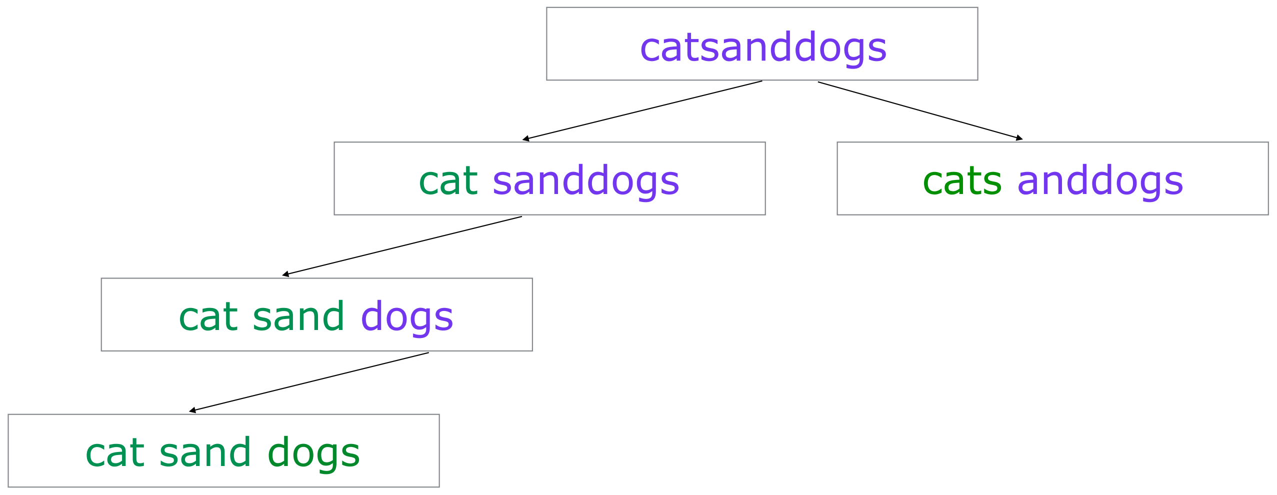
Backtracking

Input = "catsanddog"
Dict = ["cat", "cats", "and", "sand", "dog"]



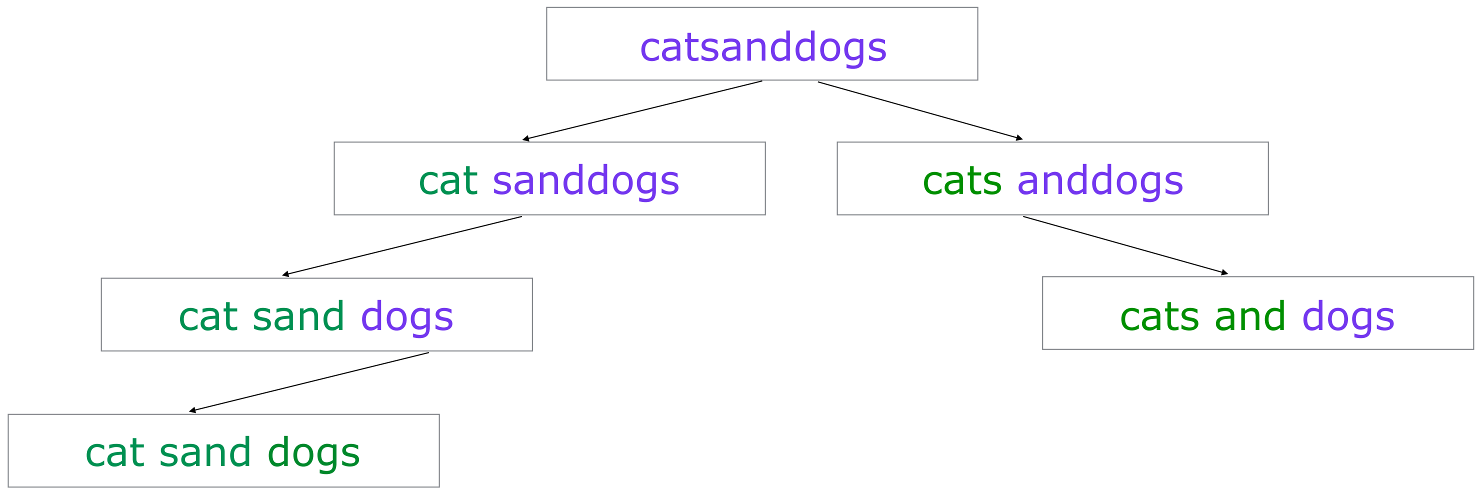
Backtracking

Input = "catsanddog"
Dict = ["cat", "cats", "and", "sand", "dog"]



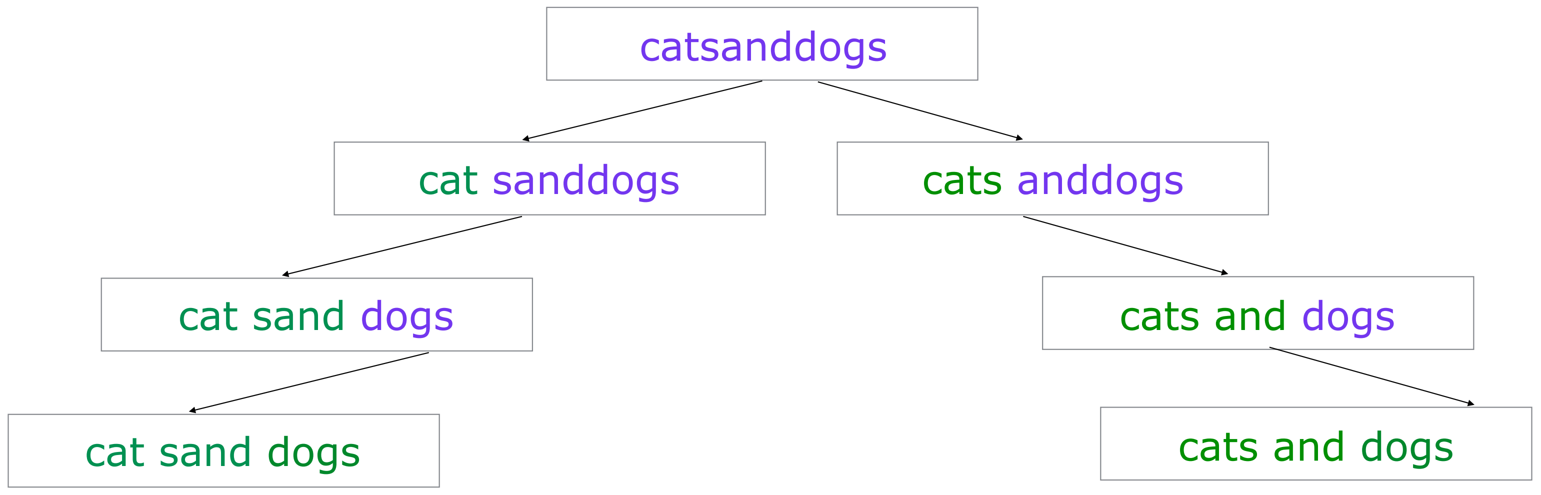
Backtracking

Input = "catsanddog"
Dict = ["cat", "cats", "and", "sand", "dog"]



Backtracking

Input = "catsanddog"
Dict = ["cat", "cats", "and", "sand", "dog"]



Java

```java
public static void wordBreak(String input, HashSet<String>
dict, List<String> partial){
    if(input.length() == 0){
        System.out.println(Arrays.toString(partial.toArray()));
        return;
    }
    for(int i=0;i<input.length();i++){
        String word = input.substring(0,i+1);
        if(dict.contains(word)){
            partial.add(word);
            wordBreak(input.substring(i+1),dict,partial);
            partial.remove(partial.size()-1);
        }
    }
}
```

Python

```python
def word_break(input_string, partial, dictionary={}):
    if len(input_string) == 0:
        print(partial)
        return

    for i in range(0, len(input_string)):
        word = input_string[:i+1]
        if word in dictionary:
            partial.append(word)
            word_break(input_string[i+1:], partial, dictionary)
            partial.pop()
```