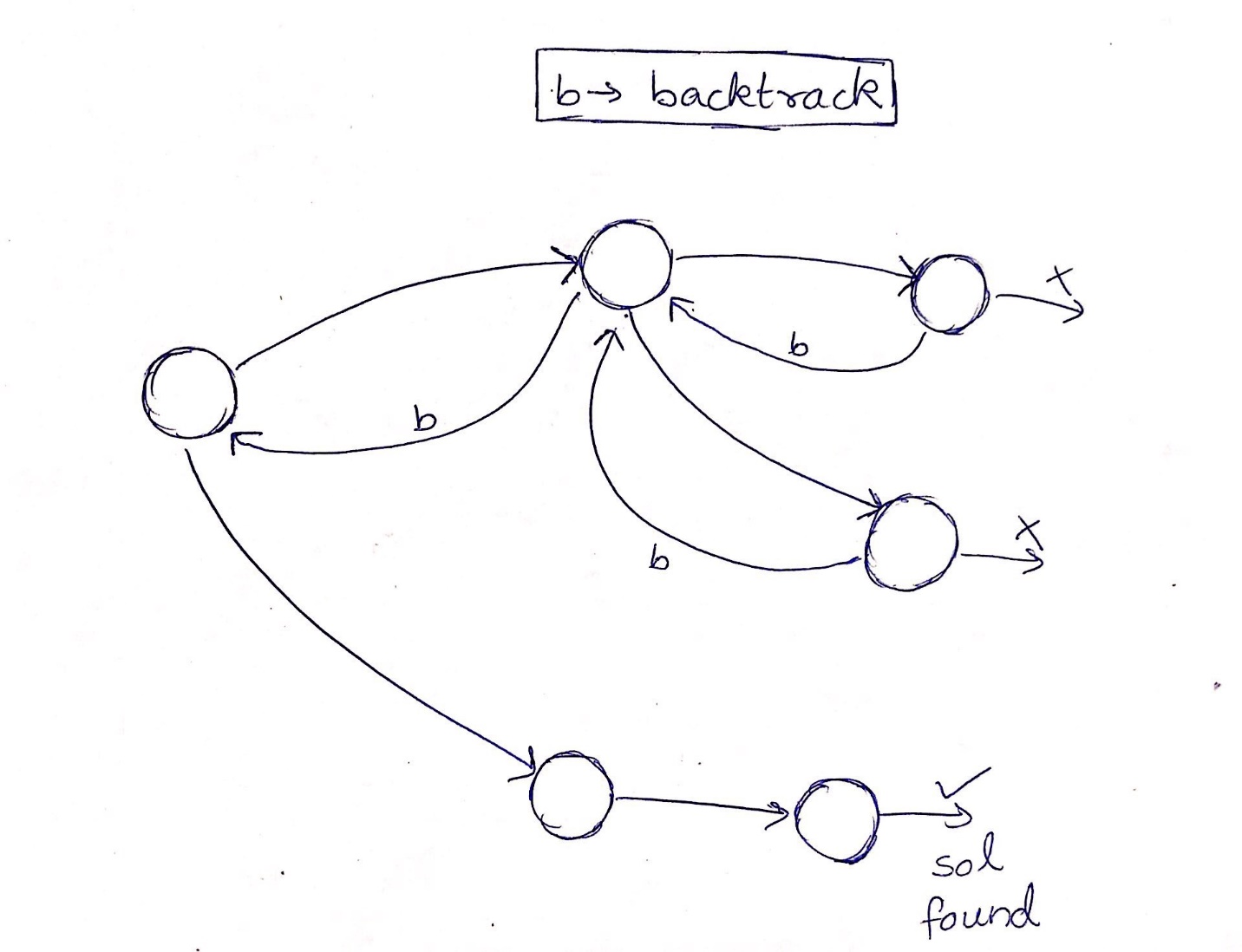
Backtracking

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. 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).

**Problem Statement**[#](https://www.educative.io/courses/grokking-the-coding-interview/gx2OqlvEnWG#problem-statement)

Given a set with distinct elements, find all of its distinct subsets.

**Example 1:**

Input: [1, 3]  
Output: [], [1], [3], [1,3]

**Example 2:**

Input: [1, 5, 3]  
Output: [], [1], [5], [3], [1,5], [1,3], [5,3], [1,5,3]

### Try

import java.util.\*;

class Subsets {

  public static List<List<Integer>> findSubsets(int[] nums) {

    List<List<Integer>> subsets = new ArrayList<>();

    // TODO: Write your code here

    return subsets;

  }

  public static void main(String[] args) {

    List<List<Integer>> result = Subsets.findSubsets(new int[] { 1, 3 });

    System.out.println("Here is the list of subsets: " + result);

    result = Subsets.findSubsets(new int[] { 1, 5, 3 });

    System.out.println("Here is the list of subsets: " + result);

  }

}

Solution:

import java.util.\*;

class Subsets {

  public static List<List<Integer>> findSubsets(int[] nums) {

    List<List<Integer>> subsets = new ArrayList<>();

    // TODO: Write your code here

    return subsets;

List<List<Integer>> subsets  = new ArrayList<>();

Arrays.sort(nums);

backtrack(subsets, new ArrayList<>(), nums, 0);

return subsets ;

  }

public static  void backtrack(List<List<Integer>> list , List<Integer> tempList, int [] nums, int start){

list.add(new ArrayList<>(tempList));

for(int i = start; i < nums.length; i++){

tempList.add(nums[i]);

backtrack(list, tempList, nums, i + 1);

tempList.remove(tempList.size() - 1);

}

}

  public static void main(String[] args) {

    List<List<Integer>> result = Subsets.findSubsets(new int[] { 1, 3 });

    System.out.println("Here is the list of subsets: " + result);

    result = Subsets.findSubsets(new int[] { 1, 5, 3 });

    System.out.println("Here is the list of subsets: " + result);

  }

}

### Problem Statement [#](https://www.educative.io/courses/grokking-the-coding-interview/7npk3V3JQNr#problem-statement)

Given a set of numbers that might contain duplicates, find all of its distinct subsets.

**Example 1:**

Input: [1, 3, 3]  
Output: [], [1], [3], [1,3], [3,3], [1,3,3]

**Example 2:**

Input: [1, 5, 3, 3]  
Output: [], [1], [5], [3], [1,5], [1,3], [5,3], [1,5,3], [3,3], [1,3,3], [3,3,5], [1,5,3,3]

### Try

class SubsetWithDuplicates {

  public static List<List<Integer>> subsetsWithDup (int[] nums) {

    List<List<Integer>> subsets = new ArrayList<>();

    // TODO

    return subsets;

  }

  public static void main(String[] args) {

    List<List<Integer>> result = SubsetWithDuplicates.subsetsWithDup(new int[] { 1, 3, 3 });

    System.out.println("Here is the list of subsets: " + result);

    result = SubsetWithDuplicates.subsetsWithDup(new int[] { 1, 5, 3, 3 });

    System.out.println("Here is the list of subsets: " + result);

  }

}

### Code

class SubsetWithDuplicates {

  public static List<List<Integer>> subsetsWithDup(int[] nums) {

    List<List<Integer>> subsets = new ArrayList<>();

    // TODO

Arrays.sort(nums);

backtrack(subsets, new ArrayList<>(), nums, 0);

return subsets;

    return subsets;

  }

public static void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, int start){

list.add(new ArrayList<>(tempList));

for(int i = start; i < nums.length; i++){

if(i > start && nums[i] == nums[i-1]) continue; // skip duplicates

tempList.add(nums[i]);

backtrack(list, tempList, nums, i + 1);

tempList.remove(tempList.size() - 1);

}

}

  public static void main(String[] args) {

    List<List<Integer>> result = SubsetWithDuplicates.subsetsWithDup(new int[] { 1, 3, 3 });

    System.out.println("Here is the list of subsets: " + result);

    result = SubsetWithDuplicates.subsetsWithDup(new int[] { 1, 5, 3, 3 });

    System.out.println("Here is the list of subsets: " + result);

  }

}

### Problem Statement [#](https://www.educative.io/courses/grokking-the-coding-interview/B8R83jyN3KY#problem-statement)

Given a set of distinct numbers, find all of its permutations.

**Permutation** is defined as the re-arranging of the elements of the set. For example, {1, 2, 3} has the following six permutations:

1. {1, 2, 3}
2. {1, 3, 2}
3. {2, 1, 3}
4. {2, 3, 1}
5. {3, 1, 2}
6. {3, 2, 1}

If a set has ‘n’ distinct elements it will have n!*n*! permutations.

**Example 1:**

Input: [1,3,5]  
Output: [1,3,5], [1,5,3], [3,1,5], [3,5,1], [5,1,3], [5,3,1]

### Try

import java.util.\*;

class Permutations {

  public static List<List<Integer>> permute(int[] nums) {

    List<List<Integer>> result = new ArrayList<>();

    // TODO: Write your code here

    return result;

  }

  public static void main(String[] args) {

    List<List<Integer>> result = Permutations. permute(new int[] { 1, 3, 5 });

    System.out.print("Here are all the permutations: " + result);

  }

}

### Code

import java.util.\*;

class Permutations {

  public static List<List<Integer>> permute(int[] nums) {

    List<List<Integer>> result = new ArrayList<>();

    // TODO: Write your code here

backtrack(list, new ArrayList<>(), nums);

return result;

  }

public static  void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums){

if(tempList.size() == nums.length){

list.add(new ArrayList<>(tempList));

} else{

for(int i = 0; i < nums.length; i++){

if(tempList.contains(nums[i])) continue; // element already exists, skip

tempList.add(nums[i]);

backtrack(list, tempList, nums);

tempList.remove(tempList.size() - 1);

}

}

}

  public static void main(String[] args) {

    List<List<Integer>> result = Permutations. permute(new int[] { 1, 3, 5 });

    System.out.print("Here are all the permutations: " + result);

  }

}

### Problem Statement [#](https://www.educative.io/courses/grokking-the-coding-interview/B8R83jyN3KY#problem-statement)

Given a set of numbers that contain duplicates find all of its possible unique permutations.

.

**Example 1:**

Input: [1,1,2]  
Output: [ [1,1,2], [1,2,1], [2,1,1] ]

### Try

import java.util.\*;

class Permutations {

  public static List<List<Integer>> permuteUnique(int[] nums) {

    List<List<Integer>> result = new ArrayList<>();

    // TODO: Write your code here

    return result;

  }

  public static void main(String[] args) {

    List<List<Integer>> result = Permutations.permuteUnique(new int[] { 1, 3, 5 });

    System.out.print("Here are all the permutations: " + result);

  }

}

### Code

import java.util.\*;

class Permutations {

  public static List<List<Integer>> permuteUnique(int[] nums) {

    List<List<Integer>> result = new ArrayList<>();

    // TODO: Write your code here

Arrays.sort(nums);

backtrack(list, new ArrayList<>(), nums, new boolean[nums.length]);

return result;

  }

public static  void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, boolean [] used){

if(tempList.size() == nums.length){

list.add(new ArrayList<>(tempList));

} else{

for(int i = 0; i < nums.length; i++){

if(used[i] || i > 0 && nums[i] == nums[i-1] && !used[i - 1]) continue;

used[i] = true;

tempList.add(nums[i]);

backtrack(list, tempList, nums, used);

used[i] = false;

tempList.remove(tempList.size() - 1);

}

}

}

  public static void main(String[] args) {

    List<List<Integer>> result = Permutations.permuteUnique(new int[] { 1, 3, 5 });

    System.out.print("Here are all the permutations: " + result);

  }

}

### Problem Statement (Combination Sum)[#](https://www.educative.io/courses/grokking-the-coding-interview/B8R83jyN3KY#problem-statement)

Given a **set** of candidate numbers (candidates) **(without duplicates)** and a target number (target), find all unique combinations in candidates where the candidate numbers sums to target.

The **same** repeated number may be chosen from candidates unlimited number of times.

**Note:**

* All numbers (including target) will be positive integers.
* The solution set must not contain duplicate combinations.

**Example 1:**

**Input:** candidates = [2,3,6,7], target = 7,

**A solution set is:**

[

[7],

[2,2,3]

]

### Try

import java.util.\*;

class CombinationSum {

public static List<List<Integer>> combinationSum(int[] candidates, int target) {

List<List<Integer>> result = new ArrayList<>();

    // TODO: Write your code here

    return result;

}

  public static void main(String[] args) {

    List<List<Integer>> result = CombinationSum.combinationSum(new int[] { 2, 3, 5 },8);

    System.out.print("Here are all the permutations: " + result);

  }

}

### Code

import java.util.\*;

class CombinationSum {

public static List<List<Integer>> combinationSum(int[] candidates, int target) {

List<List<Integer>> list = new ArrayList<>();

Arrays.sort(nums);

backtrack(list, new ArrayList<>(), nums, target, 0);

return list;

}

public static  void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, int remain, int start){

if(remain < 0) return;

else if(remain == 0) list.add(new ArrayList<>(tempList));

else{

for(int i = start; i < nums.length; i++){

tempList.add(nums[i]);

backtrack(list, tempList, nums, remain - nums[i], i); // not i + 1 because we can reuse same elements

tempList.remove(tempList.size() - 1);

}

}

}

  public static void main(String[] args) {

    List<List<Integer>> result = CombinationSum.combinationSum(new int[] { 2, 3, 5 },8);

    System.out.print("Here are all the permutations: " + result);

  }

}

### Problem Statement (Combination Sum ||)[#](https://www.educative.io/courses/grokking-the-coding-interview/B8R83jyN3KY#problem-statement)

Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sums to target.

Each number in candidates may only be used **once** in the combination.

**Note:**

* All numbers (including target) will be positive integers.
* The solution set must not contain duplicate combinations.

**Example 1:**

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

**A solution set is:**

[

[1, 7],

[1, 2, 5],

[2, 6],

[1, 1, 6]

]

**Example 2:**

**Input:** candidates = [2,5,2,1,2], target = 5,

**A solution set is:**

[

  [1,2,2],

  [5]

]

### Try

import java.util.\*;

class CombinationSum2 {

public static List<List<Integer>> combinationSum2(int[] candidates, int target) {

List<List<Integer>> result = new ArrayList<>();

    // TODO: Write your code here

    return result;

}

  public static void main(String[] args) {

    List<List<Integer>> result = CombinationSum2.combinationSum2(new int[] { 2, 5,2,1,2 },5);

    System.out.print("Here are all the permutations: " + result);

  }

}

### Code

import java.util.\*;

class CombinationSum2 {

public static List<List<Integer>> combinationSum2(int[] candidates, int target) {

List<List<Integer>> list = new ArrayList<>();

Arrays.sort(nums);

backtrack(list, new ArrayList<>(), nums, target, 0);

return list;

}

public static void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, int remain, int start){

if(remain < 0) return;

else if(remain == 0) list.add(new ArrayList<>(tempList));

else{

for(int i = start; i < nums.length; i++){

if(i > start && nums[i] == nums[i-1]) continue; // skip duplicates

tempList.add(nums[i]);

backtrack(list, tempList, nums, remain - nums[i], i + 1);

tempList.remove(tempList.size() - 1);

}

}

}

  public static void main(String[] args) {

    List<List<Integer>> result = CombinationSum2.combinationSum2(new int[] { 2, 5,2,1,2 },5);

    System.out.print("Here are all the permutations: " + result);

  }

}

### Problem Statement (Combinations)[#](https://www.educative.io/courses/grokking-the-coding-interview/B8R83jyN3KY#problem-statement)

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

**Example:**

**Input:** n = 4, k = 2

**Output:**

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4],

]

### Try

import java.util.\*;

class Combination {

public static List<List<Integer>> combine(int n, int k) {

List<List<Integer>> result = new ArrayList<>();

    // TODO: Write your code here

    return result;

}

  public static void main(String[] args) {

    List<List<Integer>> result = Combination.combine(4,2);

    System.out.print("Here are all the permutations: " + result);

  }

}

### Code

import java.util.\*;

class Combination {

public static List<List<Integer>> combine(int n, int k) {

List<List<Integer>> list=new ArrayList<List<Integer>>();

backtrack(list,new ArrayList<Integer>(),n,k,1);

return list;

}

public static void backtrack(List<List<Integer>> list,ArrayList<Integer> templist,int n, int k,int start){

if(templist.size()==k)

list.add(new ArrayList<>(templist));

else if(templist.size()>k)return;

for(int i=start;i<=n;i++){

templist.add(i);

backtrack(list,templist,n,k,i+1);

templist.remove(templist.size()-1);

}

}

  public static void main(String[] args) {

    List<List<Integer>> result = Combination.combine(4,2);

    System.out.print("Here are all the permutations: " + result);

  }

}

### Problem Statement (Palindrome Partitioning)[#](https://www.educative.io/courses/grokking-the-coding-interview/B8R83jyN3KY#problem-statement)

Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

**Example:**

**Input:** "aab"

**Output:**

[

["aa","b"],

["a","a","b"]

]

### Try

import java.util.\*;

class Palindrome {

public static List<List<String>> partition(String s) {

List<List<String>> result = new ArrayList<>();

    // TODO: Write your code here

    return result;

}

  public static void main(String[] args) {

    List<List<String>> result = Palindrome.partition(“aab”);

    System.out.print("Here are all the palindrome partitioning: " + result);

  }

}

### Code

import java.util.\*;

class Palindrome {

public static List<List<String>> partition(String s) {

List<List<String>> list = new ArrayList<>();

backtrack(list, new ArrayList<>(), s, 0);

return list;

}

  public static void backtrack(List<List<String>> list, List<String> tempList, String s, int start){

if(start == s.length())

list.add(new ArrayList<>(tempList));

else{

for(int i = start; i < s.length(); i++){

if(isPalindrome(s, start, i)){

tempList.add(s.substring(start, i + 1));

backtrack(list, tempList, s, i + 1);

tempList.remove(tempList.size() - 1);

}

}

}

}

public static boolean isPalindrome(String s, int low, int high){

while(low < high)

if(s.charAt(low++) != s.charAt(high--)) return false;

return true;

}

  public static void main(String[] args) {

    List<List<String>> result = Palindrome.partition(“aab”);

    System.out.print("Here are all the palindrome partitioning: " + result);

  }

}