Backtracking

public class Solution {

public boolean isMatch(String s, String p) {

//Java note: s.substring(n) will be "" if n == s.length(), but if n > s.length(), index oob error

int i = 0, j = 0;

//you don't have to construct a state machine for this problem

if (s.length() == 0) {

return checkEmpty(p);

}

if (p.length() == 0) {

return false;

}

char c1 = s.charAt(0);

char d1 = p.charAt(0), d2 = '0'; //any init value except '\*'for d2 will do

if (p.length()>1){

d2 = p.charAt(1);

}

if (d2 == '\*') {

if (compare(c1, d1)) {

//fork here: 1. consume the character, and use the same pattern again.

//2. keep the character, and skip 'd1\*' pattern

//Here is also an opportunity to use DP, but the idea is the same

return isMatch(s.substring(1), p) || isMatch(s, p.substring(2));

}

else {

return isMatch(s, p.substring(2));

}

}

else {

if (compare(c1, d1)) {

return isMatch(s.substring(1), p.substring(1));

}

else {

return false;

}

}

}

public boolean compare(char c1, char d1){

return d1 == '.' || c1 == d1;

}

public boolean checkEmpty(String p) {

if (p.length()%2 != 0) {

return false;

}

for (int i = 1; i < p.length(); i+=2) {

if (p.charAt(i) != '\*') {

return false;

}

}

return true;

}

}

22 Generate Parentheses ---- ---- ---- DFS (subset) ---- ---- ---- Medium

Given *n* pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given *n* = 3, a solution set is:

[

"((()))",

"(()())",

"(())()",

"()(())",

"()()()"

]

public class Solution {

public ArrayList<String> generateParenthesis(int n) {

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

if (n <= 0) {

return result;

}

helper(result, "", n, n);

return result;

}

public void helper(ArrayList<String> result,

String paren, // current paren

int left, // how many left paren we need to add

int right) { // how many right paren we need to add

if (left == 0 && right == 0) {

result.add(paren);

return;

}

if (left > 0) {

helper(result, paren + "(", left - 1, right);

}

if (right > 0 && left < right) {

helper(result, paren + ")", left, right - 1);

}

}

}

46. Permutations ---- DFS(Subset) ---- Medium

Given a collection of **distinct** numbers, return all possible permutations.

For example,  
[1,2,3] have the following permutations:

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

Top of Form

Bottom of Form

public class Solution {

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

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

if (nums == null) {

return results;

}

if (nums.length == 0) {

results.add(new ArrayList<Integer>());

return results;

}

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

Set<Integer> set = new HashSet<>();

helper(nums, permutation, set, results);

return results;

}

public void helper(int[] nums,

List<Integer> permutation,

Set<Integer> set,

List<List<Integer>> results) {

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

results.add(new ArrayList<Integer>(permutation));

return;

}

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

if (set.contains(nums[i])) {

continue;

}

permutation.add(nums[i]);

set.add(nums[i]);

helper(nums, permutation, set, results);

set.remove(nums[i]);

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

}

}

}

39 Combination Sum ---- DFS(Subset) ---- Medium

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

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

Note:

All numbers (including target) will be positive integers.

The solution set must not contain duplicate combinations.

For example, given candidate set [2, 3, 6, 7] and target 7,

A solution set is:

[

[7],

[2, 2, 3]

]

public class Solution {

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

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

if (candidates == null) {

return result;

}

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

Arrays.sort(candidates);

helper(candidates, 0, target, combination, result);

return result;

}

void helper(int[] candidates,

int index,

int target,

List<Integer> combination,

List<List<Integer>> result) {

if (target == 0) {

result.add(new ArrayList<Integer>(combination));

return;

}

for (int i = index; i < candidates.length; i++) {

if (candidates[i] > target) {

break;

}

if (i != index && candidates[i] == candidates[i - 1]) {

continue;

}

combination.add(candidates[i]);

helper(candidates, i, target - candidates[i], combination, result);

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

}

}

}

77. Combination ---- DFS(subset) ---- Medium

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

For example,

If n = 4 and k = 2, a solution is:

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4],

]

public class Solution {

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

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

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

helper(rst, solution, n, k, 1);

return rst;

}

private void helper(List<List<Integer>> rst, List<Integer> solution,

int n, int k, int start) {

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

rst.add(new ArrayList(solution));

return;

}

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

solution.add(i);

// the new start should be after the next number after i

helper(rst, solution, n, k, i + 1);

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

}

}

}

93. Restore IP Addresses

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example:  
Given "25525511135",

return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

public class Solution {

public ArrayList<String> restoreIpAddresses(String s) {

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

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

if(s.length() <4 || s.length() > 12)

return result;

helper(result, list, s , 0);

return result;

}

public void helper(ArrayList<String> result, ArrayList<String> list, String s, int start){

if(list.size() == 4){

if(start != s.length())

return;

StringBuffer sb = new StringBuffer();

for(String tmp: list){

sb.append(tmp);

sb.append(".");

}

sb.deleteCharAt(sb.length()-1);

result.add(sb.toString());

return;

}

for(int i=start; i<s.length() && i < start+3; i++){

String tmp = s.substring(start, i+1);

if(isvalid(tmp)){

list.add(tmp);

helper(result, list, s, i+1);

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

}

}

}

private boolean isvalid(String s){

if(s.charAt(0) == '0')

return s.equals("0"); // to eliminate cases like "00", "10"

int digit = Integer.valueOf(s);

return digit >= 0 && digit <= 255;

}

}

131 Palindrome Partitioning ---- DFS (subset) ---- Medium

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

Return all possible palindrome partitioning of s.

For example, given s = "aab",

Return

[

["aa","b"],

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

]

public class Solution {

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

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

if (s == null || s.length() == 0) {

return results;

}

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

helper(s, 0, partition, results);

return results;

}

private void helper(String s,

int startIndex,

List<String> partition,

List<List<String>> results) {

if (startIndex == s.length()) {

results.add(new ArrayList<String>(partition));

return;

}

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

String subString = s.substring(startIndex, i + 1);

if (!isPalindrome(subString)) {

continue;

}

partition.add(subString);

helper(s, i + 1, partition, results);

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

}

}

private boolean isPalindrome(String s) {

for (int i = 0, j = s.length() - 1; i < j; i++, j--) {

if (s.charAt(i) != s.charAt(j)) {

return false;

}

}

return true;

}

}

class Solution {

public List<String> removeInvalidParentheses(String s) {

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

if(s == null || s.length() == 0){

result.add("");

return result;

}

Queue<String> queue = new LinkedList<>();

HashSet<String> set = new HashSet<>();

queue.offer(s);

set.add(s);

boolean found = false;

while(!queue.isEmpty()){

String ss = queue.poll();

if(isValid(ss)){

result.add(ss);

found = true;

continue;

}

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

char c = ss.charAt(i);

if(c != '(' && c != ')'){

continue;

}

String newString = ss.substring(0, i) + ss.substring(i + 1);

if(!set.contains(newString)){

queue.offer(newString);

set.add(newString);

}

}

}

return result;

}

private boolean isValid(String s){

int count = 0;

for(char c : s.toCharArray()){

if(c == '('){

count++;

}

if(c == ')' && count-- == 0){

return false;

}

}

return count == 0;

}

}