Lecture 17 – Recursion, Backtracking

1. <https://leetcode.com/problems/subsets/>

class Solution {

void allSubsets(vector<int>&nums, vector<vector<int>>&res, int i, vector<int>&path)

{

// base case

if(i==nums.size())

{

res.push\_back(path);

return;

}

// recursive case

// 1. include

path.push\_back(nums[i]);

allSubsets(nums, res, i+1, path);

path.pop\_back();

// 2. exclude

allSubsets(nums, res, i+1, path);

}

public:

vector<vector<int>> subsets(vector<int>& nums) {

vector<vector<int>> res;

vector<int> path;

allSubsets(nums, res, 0, path);

return res;

}

};

1. <https://leetcode.com/problems/permutations/>

class Solution {

void allPermute(vector<int>&nums, int i, vector<vector<int>>&res)

{

// base case

if(i==nums.size())

{

res.push\_back(nums);

return;

}

for(int j=i; j<nums.size(); j++)

{

swap(nums[i], nums[j]);

allPermute(nums, i+1, res);

// restore the array

swap(nums[i], nums[j]);

}

}

public:

vector<vector<int>> permute(vector<int>& nums) {

vector<vector<int>> res;

allPermute(nums, 0, res);

return res;

}

};

1. <https://leetcode.com/problems/letter-combinations-of-a-phone-number/>

class Solution {

void phone(string digits, vector<string>&res, string mappings[], string temp, int index)

{

// base case

if(index==digits.size())

{

res.push\_back(temp);

return;

}

string val = mappings[digits[index]-'0'];

for(int i=0; i<val.size(); i++)

{

phone(digits, res, mappings, temp+val[i], index+1);

}

}

public:

vector<string> letterCombinations(string digits) {

vector<string> res;

if(digits.size()==0)

{

return res;

}

string mappings[10] = {"", "", "abc", "def", "ghi", "jkl", "mno",

"pqrs", "tuv", "wxyz"};

phone(digits, res, mappings, "", 0);

return res;

}

};