**Day-9**

**Back Tracking**

**Student Name: Kamalpreet Singh UID: 22BCS11720**

**Branch: BE - CSE Section/Group: FL\_IOT-603-A**

**Semester: 5th Date of Performance:28-12-24**

# Generate Numbers with a Given Sum

Generate all numbers of length n whose digits sum up to a target value sum, The digits of the number will be between 0 and 9, and we will generate combinations of digits such that their sum equals the target.

**Example 1:**

**Input: n = 2 and sum = 5 Output: 14 23 32 41 50**

**Constraints:**

**1 <= n <= 9: The number of digits must be between 1 and 9.**

**1 <= sum <= 100: The sum of the digits must be between 1 and 100. The first digit cannot be zero if n > 1.**

**CODE:**

#include <iostream> #include <vector> using namespace std;

void generateNumbers(int n, int sum, string current, vector<string> &result) { if (n == 0 && sum == 0) {

result.push\_back(current); return;

}

if (n == 0 || sum < 0) return;

int start = current.empty() ? 1 : 0; for (int i = start; i <= 9; ++i) {

generateNumbers(n - 1, sum - i, current + to\_string(i), result);

}

}

int main() {

int n = 2, sum = 5; vector<string> result;

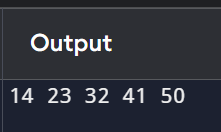
generateNumbers(n, sum, "", result); for (const string &num : result) {

cout << num << " ";

}

return 0;

}

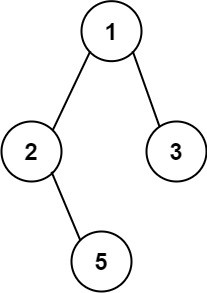


***Easy***

# Binary Tree Paths

Given the root of a binary tree, return all root-to-leaf paths in any order. A leaf is a node with no children.

**Example 1:**



**Input: root = [1,2,3,null,5]Output: ["1->2->5","1->3"] Example 2:**

**Input: root = [1]Output: ["1"]**

**Constraints:**

**The number of nodes in the tree is in the range [1, 100].**

**-100 <= Node.val <= 100 CODE:**

#include <iostream> #include <vector> #include <string> using namespace std;

struct TreeNode { int val;

TreeNode \*left, \*right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

void dfs(TreeNode \*root, string path, vector<string> &paths) { if (!root) return;

path += to\_string(root->val);

if (!root->left && !root->right) { paths.push\_back(path);

return;

}

path += "->";

dfs(root->left, path, paths); dfs(root->right, path, paths);

}

vector<string> binaryTreePaths(TreeNode \*root) { vector<string> paths;

dfs(root, "", paths); return paths;

}

int main() {

TreeNode \*root = new TreeNode(1); root->left = new TreeNode(2);

root->right = new TreeNode(3);

root->left->right = new TreeNode(5);

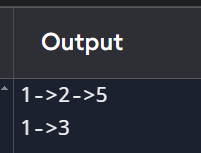
vector<string> result = binaryTreePaths(root); for (const string &path : result) {

cout << path << endl;

}

return 0;

}



***Medium:***

# Combinations

Given two integers n and k, return all possible combinations of k numbers chosen from the range [1, n].

You may return the answer in any order.

**Example 1:**

**Input: n = 4, k = 2**

**Output: [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]**

**Explanation: There are 4 choose 2 = 6 total combinations.**

**Note that combinations are unordered, i.e., [1,2] and [2,1] are considered to be the same combination.**

**Example 2:**

**Input: n = 1, k = 1 Output: [[1]]**

**Explanation: There is 1 choose 1 = 1 total combination. Constraints:**

**1 <= n <= 20**

**1 <= k <= n CODE:**

#include <iostream> #include <vector> using namespace std;

void combineHelper(int start, int n, int k, vector<int> &current, vector<vector<int>> &result) { if (k == 0) {

result.push\_back(current); return;

}

for (int i = start; i <= n; ++i) { current.push\_back(i);

combineHelper(i + 1, n, k - 1, current, result); current.pop\_back();

}

}

vector<vector<int>> combine(int n, int k) { vector<vector<int>> result;

vector<int> current;

combineHelper(1, n, k, current, result); return result;

}

int main() {

int n = 4, k = 2;

vector<vector<int>> result = combine(n, k); for (const auto &comb : result) {

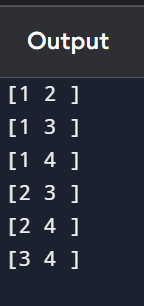
cout << "[";

for (int num : comb) cout << num << " "; cout << "]" << endl;

}

return 0;

}



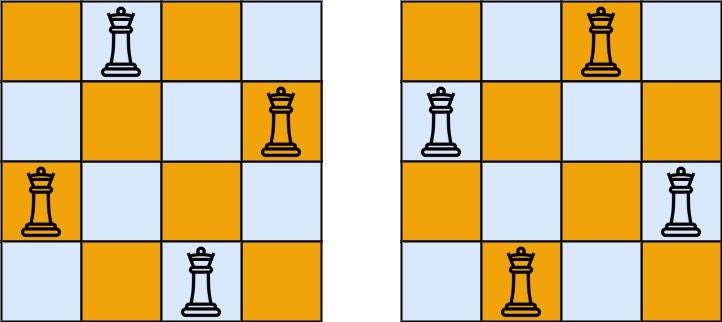
***Hard:***

# [N-Queens II](https://leetcode.com/problems/n-queens-ii/)

The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.

Given an integer n, return the number of distinct solutions to the n-queens puzzle.

**Example 1:**

****

**Input: n = 4 Output: 2**

**Explanation: There are two distinct solutions to the 4-queens puzzle as shown.**

**Example 2: Input: n = 1 Output: 1**

**Constraints:**

**1 <= n <= 9 CODE:**

#include <iostream> #include <vector> using namespace std;

void solve(int row, int n, vector<int> &cols, vector<int> &diags1, vector<int> &diags2, int &count) {

if (row == n) {

++count; return;

}

for (int col = 0; col < n; ++col) {

if (cols[col] || diags1[row - col + n - 1] || diags2[row + col]) continue; cols[col] = diags1[row - col + n - 1] = diags2[row + col] = 1; solve(row + 1, n, cols, diags1, diags2, count);

cols[col] = diags1[row - col + n - 1] = diags2[row + col] = 0;

}

}

int totalNQueens(int n) {

vector<int> cols(n, 0), diags1(2 \* n - 1, 0), diags2(2 \* n - 1, 0); int count = 0;

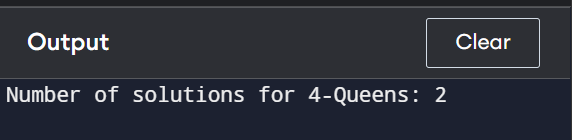
solve(0, n, cols, diags1, diags2, count); return count;

}

int main() { int n = 4;

cout << "Number of solutions for " << n << "-Queens: " << totalNQueens(n) << endl; return 0;

}



***Very Hard:***

# [Word Ladder II](https://leetcode.com/problems/word-ladder-ii/)

A transformation sequence from word beginWord to word endWord using a

dictionary wordList is a sequence of words beginWord -> s1 -> s2 -> ... -> sk such that: Every adjacent pair of words differs by a single letter.

Every si for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList. sk == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return all the shortest transformation sequences from beginWord to endWord, or an empty list if no such sequence exists. Each sequence should be returned as a list of the words [beginWord, s1, s2, ..., sk].

**Example 1:**

**Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]**

**Output: [["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]] Explanation: There are 2 shortest transformation sequences:**

**"hit" -> "hot" -> "dot" -> "dog" -> "cog"**

**"hit" -> "hot" -> "lot" -> "log" -> "cog"**

**Example 2:**

**Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"] Output: []**

**Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.**

**Constraints:**

**1 <= beginWord.length <= 5 endWord.length == beginWord.length 1 <= wordList.length <= 500**

**wordList[i].length == beginWord.length**

**beginWord, endWord, and wordList[i] consist of lowercase English letters. beginWord != endWord**

**All the words in wordList are unique.**

**The sum of all shortest transformation sequences does not exceed 105. CODE:**

#include <iostream> #include <vector> #include <unordered\_set> #include <queue>

using namespace std;

vector<vector<string>> findLadders(string beginWord, string endWord, vector<string> &wordList) {

unordered\_set<string> dict(wordList.begin(), wordList.end()); vector<vector<string>> result;

if (dict.find(endWord) == dict.end()) return result;

queue<vector<string>> paths; paths.push({beginWord});

int level = 1, minLevel = INT\_MAX; unordered\_set<string> visited;

while (!paths.empty()) { vector<string> path = paths.front();

paths.pop();

if (path.size() > level) {

for (const string &word : visited) dict.erase(word); visited.clear();

level = path.size();

if (level > minLevel) break;

}

string last = path.back();

for (int i = 0; i < last.size(); ++i) { string next = last;

for (char c = 'a'; c <= 'z'; ++c) { next[i] = c;

if (!dict.count(next)) continue; visited.insert(next); vector<string> newPath = path; newPath.push\_back(next);

if (next == endWord) { result.push\_back(newPath); minLevel = level;

} else {

paths.push(newPath);

}

}

}

}

return result;

}

int main() {

string beginWord = "hit", endWord = "cog";

vector<string> wordList = {"hot", "dot", "dog", "lot", "log", "cog"}; vector<vector<string>> result = findLadders(beginWord, endWord, wordList); for (const auto &path : result) {

for (const string &word : path) { cout << word << " ";

}

cout << endl;

}

return 0;

}

