DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: Tamanna Gupta **UID: 22BCS14867**
Branch: BE-CSE::CS201 **Section/Group: 22BCS_FL_IOT-603/B**
Semester: 5th

➢ DAY-9 [28-12-2024]

1. Generate Numbers with a Given Sum *(Very Easy)*

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, sum = 5

Output: 14 23 32 41 50

Example 2: Input: n = 3, sum = 5

Output: 104 113 122 131 140 203 212 221 230 302 311 320 401 410 500

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.

Implementation/Code:

```
#include <iostream>
#include <vector>
#include <string>
```

```
using namespace std;

class Solution {
public:
    void findNumbers(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;

        for (int digit = (current.empty() ? 1 : 0); digit <= 9; digit++) {
            findNumbers(n - 1, sum - digit, current + to_string(digit), result);
        }
    }

    vector<string> generateNumbers(int n, int sum) {
        vector<string> result;
        findNumbers(n, sum, "", result);
        return result;
    }
};

int main()
{
    Solution sol;
    vector<string> result = sol.generateNumbers(2, 5);
    for (string num : result) cout << num << " ";
    cout << endl;

    result = sol.generateNumbers(3, 5);
    for (string num : result) cout << num << " ";
    cout << endl;

    return 0;
}
```

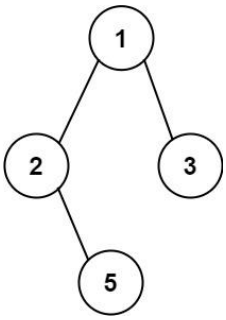
Output:

```
input
14 23 32 41 50
104 113 122 131 140 203 212 221 230 302 311 320 401 410 500


...Program finished with exit code 0
Press ENTER to exit console.
```

2. Binary Tree Paths *(Easy)*

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

Implementation/Code:

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    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;
        }
        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);
```

```
    Solution sol;
    vector<string> paths = sol.binaryTreePaths(root);
    for (string path : paths) cout << path << endl;

    return 0;
}
```

Output:

3. Combinations *(Medium)*

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:

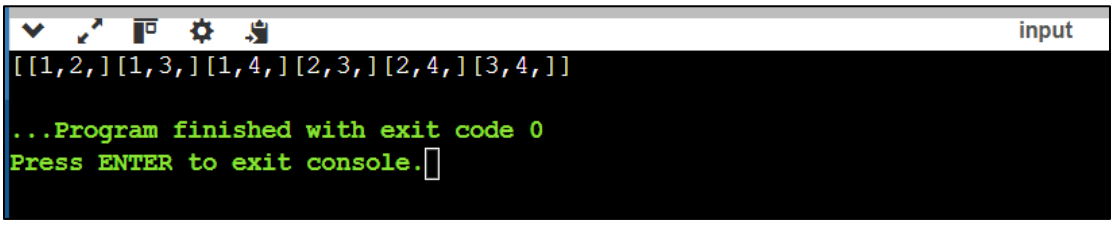
1 <= n <= 20
1 <= k <= n

Implementation/Code:

```
#include <iostream>
#include <vector>

using namespace std;

class Solution {
public:
    void combineHelper(int n, int k, int start, vector<int>& current, vector<vector<int>>&
result) {
        if (current.size() == k) {
            result.push_back(current);
            return;
        }
        for (int i = start; i <= n; i++) {
            current.push_back(i);
            combineHelper(n, k, i + 1, current, result);
            current.pop_back();
        }
    }

    vector<vector<int>> combine(int n, int k) {
        vector<vector<int>> result;
        vector<int> current;
        combineHelper(n, k, 1, current, result);
        return result;
    }
};

int main() {
    Solution sol;
    vector<vector<int>> result = sol.combine(4, 2);
    cout<<"[";
    for (auto combination : result) {
        cout<<"[";
```

```
        for (int num : combination) cout << num << ",";
        cout <<"]";
    }
    cout<<"]";

    return 0;
}
```

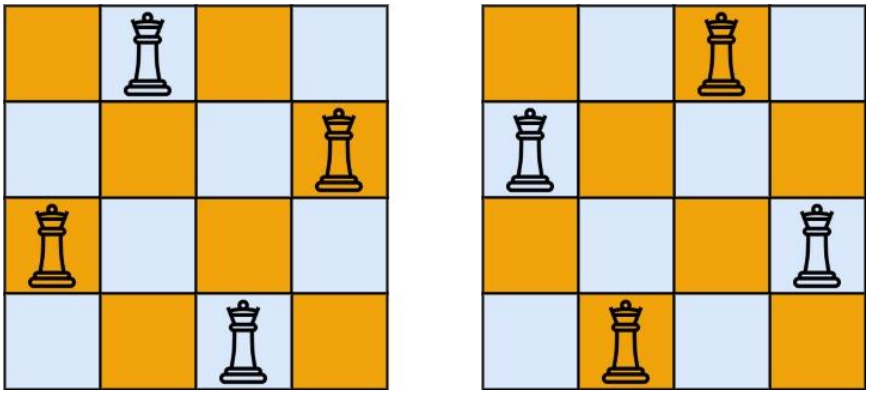
Output:



4. N-Queens II *(Hard)*

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

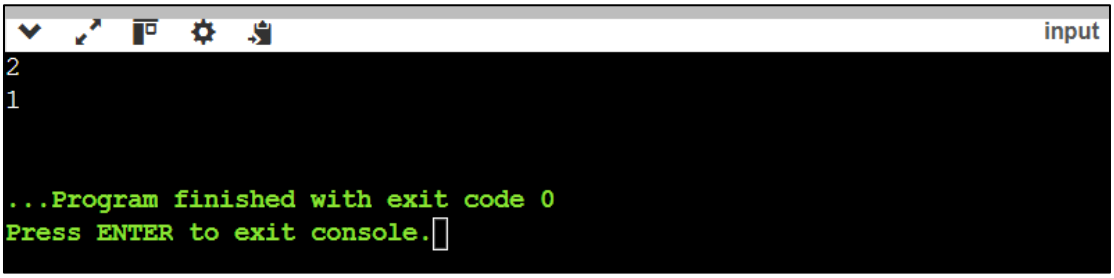
Constraints: 1 <= n <= 9

Implementation/Code:

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

class Solution {
public:
    int totalNQueens(int n) {
        vector<int> queens(n, -1);
        return backtrack(0, queens, n);
    }

private:
    int backtrack(int row, vector<int>& queens, int n) {
        if (row == n) return 1;
        int count = 0;
        for (int col = 0; col < n; col++) {
            if (isValid(queens, row, col)) {
                queens[row] = col;
                count += backtrack(row + 1, queens, n);
                queens[row] = -1;
            }
        }
        return count;
    }

    bool isValid(vector<int>& queens, int row, int col) {
        for (int i = 0; i < row; i++) {
            if (queens[i] == col || abs(queens[i] - col) == abs(i - row)) return false;
        }
```

```
        return true;
    }
};

int main() {
    Solution sol;
    cout << sol.totalNQueens(4) << endl;
    cout << sol.totalNQueens(1) << endl;
    return 0;
}
```

Output:



5. Word Ladder II *(Very Hard)*

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.

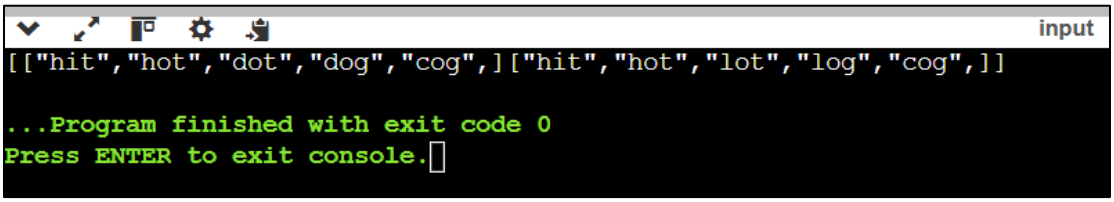
Implementation/Code:

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_set>
#include <queue>
#include <unordered_map>

using namespace std;
```

```
class Solution {
public:
    vector<vector<string>> findLadders(string beginWord, string endWord,
vector<string>& wordList) {
        unordered_set<string> wordSet(wordList.begin(), wordList.end());
        vector<vector<string>> result;
        if (wordSet.find(endWord) == wordSet.end()) return result;

        unordered_map<string, vector<string>> graph;
        unordered_map<string, int> levels;
        buildGraph(beginWord, endWord, wordSet, graph, levels);

        vector<string> path;
        backtrack(beginWord, endWord, graph, levels, path, result);
        return result;
    }

private:
    void buildGraph(string& beginWord, string& endWord, unordered_set<string>&
wordSet,
                    unordered_map<string, vector<string>>& graph, unordered_map<string,
int>& levels) {
        queue<string> q;
        q.push(beginWord);
        levels[beginWord] = 0;
        while (!q.empty()) {
            string word = q.front();
            q.pop();
            string temp = word;
            for (int i = 0; i < word.size(); i++) {
                char original = word[i];
                for (char c = 'a'; c <= 'z'; c++) {
                    word[i] = c;
                    if (wordSet.find(word) != wordSet.end() && levels.find(word) ==
levels.end()) {
                        graph[temp].push_back(word);
                        levels[word] = levels[temp] + 1;
                        q.push(word);
```

```
                } else if (levels.find(word) != levels.end() && levels[word] == levels[temp]
+ 1) {
                    graph[temp].push_back(word);
                }
            }
            word[i] = original;
        }
    }
}
void backtrack(string& word, string& endWord, unordered_map<string,
vector<string>>& graph,
               unordered_map<string, int>& levels, vector<string>& path,
vector<vector<string>>& result) {
    path.push_back(word);
    if (word == endWord) {
        result.push_back(path);
    } else {
        for (string& neighbor : graph[word]) {
            backtrack(neighbor, endWord, graph, levels, path, result);
        }
    }
    path.pop_back();
}
};

int main() {
    Solution sol;
    vector<string> wordList = {"hot", "dot", "dog", "lot", "log", "cog"};
    vector<vector<string>> result = sol.findLadders("hit", "cog", wordList);
    cout<<"[";
    for (auto path : result) {
        cout<<"[";
        for (string word : path) cout <<""<<word<<""<< ",";
        cout <<"]";
    }
    cout<<"]";
    return 0;
}
```

Output:


input
[["hit","hot","dot","dog","cog",]["hit","hot","lot","log","cog",]]
...Program finished with exit code 0
Press ENTER to exit console.