DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: Gurnoor Oberoi **UID: 22BCS15716**
Branch: BE-CSE::CS201 **Section/Group: 22BCS_FL_IOT-603/B**
Semester: 5th

➢ DAY-8 [27-12-2024]

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.

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 (current.length() == n) {
            if (sum == 0) {
                result.push_back(current);
            }
            return;
        }
        for (int digit = 0; digit <= 9; ++digit) {
            if (sum >= digit) {
                findNumbers(n, sum - digit, current + char(digit + '0'), result);
            }
        }
    }
    vector<string> generateNumbers(int n, int sum) {
```

```
        vector<string> result;
        findNumbers(n, sum, "", result);
        return result;
    }
};
int main() {
    int n, sum;
    cout << "Enter the length of the number (n): ";
    cin >> n;
    cout << "Enter the target sum: ";
    cin >> sum;
    Solution solution;
    vector<string> numbers = solution.generateNumbers(n, sum);
    cout << "Generated numbers are:" << endl;
    for (const string& number : numbers) {
        cout << number << endl;
    }
    return 0;
}
```

Output:

```
Enter the length of the number (n): 2
Enter the target sum: 5
Generated numbers are:
05
14
23
32
41
50
```

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.

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, vector<string>& result, string currentPath) {
        if (!root) return;
        currentPath += to_string(root->val);
        if (!root->left && !root->right) {
            result.push_back(currentPath);
            return;
        }
        if (root->left) dfs(root->left, result, currentPath + "->");
        if (root->right) dfs(root->right, result, currentPath + "->");
    }
    vector<string> binaryTreePaths(TreeNode* root) {
        vector<string> result;
        dfs(root, result, "");
        return result;
    }
};
int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->right = new TreeNode(5);
    Solution solution;
    vector<string> paths = solution.binaryTreePaths(root);
    cout << "All root-to-leaf paths:" << endl;
    for (const string& path : paths) {
        cout << path << endl;
    }
    return 0;
}
```

Output:

```
All root-to-leaf paths:
1->2->5
1->3
```

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.

Implementation/Code:

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    void backtrack(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);
            backtrack(n, k, i + 1, current, result);
            current.pop_back();
        }
    }
    vector<vector<int>> combine(int n, int k) {
        vector<vector<int>> result;
        vector<int> current;
        backtrack(n, k, 1, current, result);
        return result;
    }
};
void printCombinations(const vector<vector<int>>& combinations) {
    for (const auto& combination : combinations) {
        for (int num : combination) {
            cout << num << " ";
```

```
        }
        cout << endl;
    }
}
int main() {
    int n, k;
    cout << "Enter n: ";
    cin >> n;
    cout << "Enter k: ";
    cin >> k;
    Solution solution;
    vector<vector<int>> combinations = solution.combine(n, k);
    cout << "All combinations of " << k << " numbers chosen from 1 to " << n << " are:"
<< endl;
    printCombinations(combinations);
    return 0;
}
```

Output:

```
Enter n: 4
Enter k: 2
All combinations of 2 numbers chosen from 1 to 4 are:
1 2
1 3
1 4
2 3
2 4
3 4
```

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.

Implementation/Code:

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    int totalNQueens(int n) {
```

```
        int count = 0;
        vector<bool> columns(n, false);
        vector<bool> diag1(2 * n - 1, false);
        vector<bool> diag2(2 * n - 1, false);
        backtrack(n, 0, columns, diag1, diag2, count);
        return count;
    }
private:
    void backtrack(int n, int row, vector<bool>& columns, vector<bool>& diag1, vector<bool>& diag2, int& count) {
        if (row == n) {
            count++;
            return;
        }
        for (int col = 0; col < n; ++col) {
            if (!columns[col] && !diag1[row - col + (n - 1)] && !diag2[row + col]) {
                columns[col] = true;
                diag1[row - col + (n - 1)] = true;
                diag2[row + col] = true;
                backtrack(n, row + 1, columns, diag1, diag2, count);
                columns[col] = false;
                diag1[row - col + (n - 1)] = false;
                diag2[row + col] = false;
            }
        }
    }
};
int main() {
    int n;
    cout << "Enter the size of the chessboard (n): ";
    cin >> n;
    Solution solution;
    int result = solution.totalNQueens(n);
    cout << "The number of distinct solutions for the " << n << "-Queens puzzle is: " << result << endl;
    return 0;
}
```

Output:

```
Enter the size of the chessboard (n): 4
The number of distinct solutions for the 4-Queens puzzle is: 2
```

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].

Implementation/Code:

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <queue>
#include <string>
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;
        }
        queue<vector<string>> q;
        q.push({beginWord});
        unordered_map<string, bool> visited;
        visited[beginWord] = true;
        bool foundEnd = false;
```

```
        while (!q.empty() && !foundEnd) {
            unordered_map<string, bool> localVisited;
            int size = q.size();

            for (int i = 0; i < size; ++i) {
                vector<string> path = q.front();
                q.pop();
                string word = path.back();
                for (int j = 0; j < word.length(); ++j) {
                    string newWord = word;
                    for (char c = 'a'; c <= 'z'; ++c) {
                        newWord[j] = c;
                        if (wordSet.find(newWord) != wordSet.end() && !visited[newWord]) {
                            if (newWord == endWord) {
                                foundEnd = true;
                                path.push_back(newWord);
                                result.push_back(path);
                                path.pop_back();
                            } else {
                                path.push_back(newWord);
                                q.push(path);
                                localVisited[newWord] = true;
                                path.pop_back();
                            }
                        }
                    }
                }
            }
            for (auto& entry : localVisited) {
                visited[entry.first] = true;
            }
        }
        return result;
    }
};

int main() {
    Solution solution;
```

```
    string beginWord, endWord;
    int n;
    cout << "Enter the begin word: ";
    cin >> beginWord;
    cout << "Enter the end word: ";
    cin >> endWord;
    cout << "Enter the number of words in the word list: ";
    cin >> n;
    vector<string> wordList(n);
    cout << "Enter the words in the word list: " << endl;
    for (int i = 0; i < n; ++i) {
        cin >> wordList[i];
    }
    vector<vector<string>> result = solution.findLadders(beginWord, endWord,
wordList);
    if (result.empty()) {
        cout << "No valid transformation sequence exists." << endl;
    } else {
        cout << "Shortest transformation sequences:" << endl;
        for (const auto& path : result) {
            for (const string& word : path) {
                cout << word << " ";
            }
            cout << endl;
        }
    }
    return 0;
}
```

Output:

```
Enter the begin word: hit
Enter the end word: cog
Enter the number of words in the word list: 6
Enter the words in the word list:
hot dot dog lot log cog
Shortest transformation sequences:
hit hot dot dog cog
hit hot lot log cog
```