



DOMAIN WINTER WINNING CAMP

Student Name: Jyoti Kanwar

UID: 22BCS11559

Branch: BE-CSE

Section/Group: TPP_FL_603-A

DAY 9:

QUES 1: 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.

Solution:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
class Solution {
public:
    void backtrack(int n, int sum, int pos, string& current, vector<string>& result) {
        if (pos == n) {
            if (sum == 0) result.push_back(current);
            return;
        }
        for (int i = (pos == 0 ? 1 : 0); i <= 9; ++i) {
            if (sum - i >= 0) {
                current.push_back('0' + i);
                backtrack(n, sum - i, pos + 1, current, result);
                current.pop_back();
            }
        }
    }
}
```

```
}  
vector<string> generateNumbers(int n, int sum) {  
    vector<string> result;  
    string current;  
    backtrack(n, sum, 0, current, result);  
    return result;  
}  
};  
int main() {  
    Solution solution;  
    int n = 2, sum = 5;  
    vector<string> result = solution.generateNumbers(n, sum);  
    for (const auto& num : result) {  
        cout << num << " ";  
    }  
    return 0;  
}
```

```
14 23 32 41 50
```

QUES 2: 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.

Solution:

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

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

class Solution {
public:
    void findPaths(TreeNode* root, string path, vector<string>& result) {
        if (!root) return;
        path += to_string(root->val);
        if (!root->left && !root->right) {
            result.push_back(path);
            return;
        }
        path += "->";
        findPaths(root->left, path, result);
        findPaths(root->right, path, result);
    }
    vector<string> binaryTreePaths(TreeNode* root) {
        vector<string> result;
        findPaths(root, "", result);
        return result;
    }
};

int main() {
    Solution solution;
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
```

```
root->right = new TreeNode(3);
root->left->right = new TreeNode(5);
vector<string> paths = solution.binaryTreePaths(root);
for (const auto& path : paths) {
    cout << path << endl;
}
return 0;
}
```

```
1->2->5
1->3
```

QUES 3: 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.

Solution:

```
#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 solution;
    int n = 4, k = 2;
    vector<vector<int>> result = solution.combine(n, k);
    for (const auto& combination : result) {
        cout << "[";
        for (size_t i = 0; i < combination.size(); ++i) {
            cout << combination[i] << (i < combination.size() - 1 ? "," : "");
        }
        cout << "] ";
    }
    return 0;
}
```

```
[1,2] [1,3] [1,4] [2,3] [2,4] [3,4]
```

QUES 4: 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.

Solution:

```
#include <iostream>

#include <vector>

using namespace std;

class Solution {
public:
    void solve(int n, int row, vector<int>& columns, vector<int>& diag1, vector<int>& diag2,
int& count) {
        if (row == n) {
            count++;
            return;
        }
        for (int col = 0; col < n; ++col) {
            if (columns[col] || diag1[row + col] || diag2[row - col + n - 1]) continue;
            columns[col] = diag1[row + col] = diag2[row - col + n - 1] = 1;
            solve(n, row + 1, columns, diag1, diag2, count);
            columns[col] = diag1[row + col] = diag2[row - col + n - 1] = 0;
        }
    }
}

int totalNQueens(int n) {
    vector<int> columns(n, 0), diag1(2 * n - 1, 0), diag2(2 * n - 1, 0);
    int count = 0;
    solve(n, 0, columns, diag1, diag2, count);
    return count;
}
```

```
};

int main() {
    Solution solution;

    int n = 4;

    cout << solution.totalNQueens(n) << endl;

    return 0;
}
```

2

QUES 5: 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 s_i for $1 \leq i \leq k$ 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]`.

Solution:

```
#include <bits/stdc++.h>
```

```
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>>> parentMap;
unordered_set<string> visited;
queue<string> q;
q.push(beginWord);
visited.insert(beginWord);
bool found = false;
while (!q.empty() && !found) {
    int levelSize = q.size();
    unordered_set<string> levelVisited;
    for (int i = 0; i < levelSize; ++i) {
        string current = q.front();
        q.pop();
        for (int j = 0; j < current.size(); ++j) {
            string next = current;
            for (char c = 'a'; c <= 'z'; ++c) {
                next[j] = c;
                if (next == current) continue;
                if (next == endWord) found = true;
                if (wordSet.find(next) != wordSet.end() && visited.find(next) ==
visited.end()) {
                    levelVisited.insert(next);
                    parentMap[next].push_back(current);
                }
            }
        }
    }
    for (const string& word : levelVisited) {
        visited.insert(word);
        q.push(word);
    }
}
```



```
    }  
    vector<string> path;  
    backtrack(beginWord, endWord, parentMap, path, result);  
    return result;  
}  
  
private:  
    void backtrack(const string& current, const string& endWord,  
        unordered_map<string, vector<string>>& parentMap,  
        vector<string>& path, vector<vector<string>>& result) {  
        path.push_back(current);  
        if (current == endWord) {  
            result.push_back(vector<string>(path.rbegin(), path.rend()));  
        } else {  
            for (const string& parent : parentMap[current]) {  
                backtrack(parent, endWord, parentMap, path, result);  
            }  
        }  
        path.pop_back();  
    }  
};  
  
int main() {  
    Solution sol;  
    string beginWord = "hit", endWord = "cog";  
    vector<string> wordList = {"hot", "dot", "dog", "lot", "log", "cog"};  
    vector<vector<string>> result = sol.findLadders(beginWord, endWord, wordList);  
    for (const auto& path : result) {  
        for (const auto& word : path) {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        cout << word << " ";  
    }  
    cout << endl;  
}  
return 0;  
}  
No transformation sequence found.
```