DOMAIN WINTER CAMP WORKSHEET

DAY-9 (28/12/2024)

Student Name :- Pratham Kapoor **University ID :- 22BCS10732**

Branch :- B.E. (C.S.E.) **Section/Group :- FL_ 603-A**

Problem-1 :- 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. **(Very Easy)**

Source Code

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

void FindNumbers(int N,int Sum,string Current,vector<string>&Results){
if(N==0){
if(Sum==0&&Current[0]!='0'){
Results.push_back(Current);
}
return;
}
for(int Digit=0;Digit<=9;++Digit){
if(Sum-Digit>=0){
FindNumbers(N-1,Sum-Digit,Current+to_string(Digit),Results);
}
}
}
```

```
int main() {
int N,Sum;
cout<<"Enter the Length of the Number :- ";
cin>>N;
cout<<"Enter the Target Sum :- ";
cin>>Sum;
vector<string>Results;
FindNumbers(N,Sum,"",Results);
cout<<"\nThe Generated Numbers are ";
for(string Number:Results){
cout<<Number<<" ";
}
return 0;
}
```

Output

```
Enter the Length of the Number :- 2
Enter the Target Sum :- 5

The Generated Numbers are 14 23 32 41 50
```

Problem-2 :- Given the root of a binary tree, return all root-to-leaf paths in any order. A leaf is a node with no children. **(Easy)**

Source Code

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

struct TreeNode
{
```

```
    int Val;
    TreeNode* Left;
    TreeNode* Right;
    TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

void FindPaths(TreeNode* Node, string Path, vector<string>& Paths)
{
    if (Node == NULL)
    {
        return;
    }
    Path += to_string(Node->Val);
    if (Node->Left == NULL && Node->Right == NULL)
    {
        Paths.push_back(Path);
        return;
    }
    Path += "->";
    FindPaths(Node->Left, Path, Paths);
    FindPaths(Node->Right, Path, Paths);
}

vector<string> BinaryTreePaths(TreeNode* Root)
{
    vector<string> Paths;
    FindPaths(Root, "", Paths);
    return Paths;
}

TreeNode* BuildTree()
{
```

```
    string Val;
    cin >> Val;
    if (Val == "#") {
        return NULL;
    }
    TreeNode* Root = new TreeNode(stoi(Val));
    cout << "Enter the Left Child of " << Val << " - ";
    Root->Left = BuildTree();
    cout << "Enter the Right Child of " << Val << " - ";
    Root->Right = BuildTree();
    return Root;
}

int main() {
    cout << "Enter the Value for Root Node :- ";
    TreeNode* Root = BuildTree();
    vector<string> Paths = BinaryTreePaths(Root);

    cout << "\nThe Root-To-Leaf Paths are ";
    cout << "[";
    for (size_t i = 0; i < Paths.size(); ++i)
    {
        cout << "\"" << Paths[i] << "\"";
        if (i != Paths.size() - 1)
        {
            cout << ",";
        }
    }
    cout << "]\n";

    return 0;
}
```

Output

```
Enter the Value for Root Node :- 1
Enter the Left Child of 1 - 2
Enter the Left Child of 2 - #
Enter the Right Child of 2 - 5
Enter the Left Child of 5 - #
Enter the Right Child of 5 - #
Enter the Right Child of 1 - 3
Enter the Left Child of 3 - #
Enter the Right Child of 3 - #

The Root-To-Leaf Paths are ["1->2->5","1->3"]
```

Problem-3 :- 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. **(Medium)**

Source Code

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

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;
}

int main() {
    int n, k;
    cout << "Enter the Value of N :- ";
    cin >> n;
    cout << "Enter the Value of K :-  ";
    cin >> k;
    vector<vector<int>> result = combine(n, k);
    cout << "\nThe Combinations of " << k << " Numbers Chosen from the Range [1," << n <<"] are ";
    cout << "[";
    for(int i = 0; i < result.size(); ++i) {
        cout << "[";
        for(int j = 0; j < result[i].size(); ++j) {
            cout << result[i][j];
            if(j != result[i].size() - 1) {
                cout << ",";
            }
        }
        cout << "]";
        if(i != result.size() - 1) {
            cout << ",";
        }
    }
    cout << "]" << endl;
```

```
    return 0;
}
```

Output

```
Enter the Value of N :- 4
Enter the Value of K :-  2

All Possible Combinations are [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
```

Problem-4 :- 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. **(Hard)**

Source Code

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

int CountSolutions(int n,int row,vector<int>& cols,vector<int>& diag1,vector<int>& diag2) {
    if(row==n) {
        return 1;
    }
    int solutions=0;
    for(int col=0;col<n;++col) {
        if(cols[col]||diag1[row+col]||diag2[row-col+n-1]) {
            continue;
        }
        cols[col]=diag1[row+col]=diag2[row-col+n-1]=1;
        solutions+=CountSolutions(n,row+1,cols,diag1,diag2);
        cols[col]=diag1[row+col]=diag2[row-col+n-1]=0;
```

```
    }
    return solutions;
}

int TotalNQueens(int n) {
    vector<int> cols(n,0),diag1(2*n-1,0),diag2(2*n-1,0);
    return CountSolutions(n,0,cols,diag1,diag2);
}

int main() {
    int n;
    cout<<"Enter the Value of N :- ";
    cin>>n;
    int result=TotalNQueens(n);
    cout<<"\nThe Number of Distinct Solutions to the "<<n<<"-Queens Puzzle
are "<<result<<endl;
    return 0; }
```

Output

```
Enter the Value of N :- 4

The Number of Distinct Solutions to the 4-Queens Puzzle are 2
```

Problem-5 :- 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 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]. **(Very Hard)**

Source Code

```
#include<iostream>
#include<vector>
#include<unordered_set>
#include<unordered_map>
#include<queue>
using namespace std;

void Backtrack(string beginWord, string endWord, unordered_map<string, vector<string>>& graph, vector<string>& path, vector<vector<string>>& result) {
    path.push_back(beginWord);
    if (beginWord == endWord) {
        result.push_back(path);
    } else {
        for (const string& nextWord : graph[beginWord]) {
            Backtrack(nextWord, endWord, graph, path, result);
        }
    }
    path.pop_back();
}

vector<vector<string>> FindLadders(string beginWord, string endWord, vector<string>& wordList) {
    unordered_set<string> wordSet(wordList.begin(), wordList.end());
    unordered_map<string, vector<string>> graph;
    vector<vector<string>> result;

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

    queue<string> q;
```

```
q.push(beginWord);
wordSet.erase(beginWord);

bool found = false;
unordered_set<string> visitedLevel;

while (!q.empty() && !found) {
    unordered_set<string> visitedNextLevel;
    int size = q.size();

    for (int i = 0; i < size; ++i) {
        string word = q.front();
        q.pop();

        for (int j = 0; j < word.size(); ++j) {
            string temp = word;

            for (char c = 'a'; c <= 'z'; ++c) {
                temp[j] = c;
                if (wordSet.find(temp) != wordSet.end()) {
                    graph[word].push_back(temp);
                    if (temp == endWord) {
                        found = true;
                    }
                    if (visitedLevel.find(temp) == visitedLevel.end()) {
                        visitedNextLevel.insert(temp);
                        q.push(temp);
                    }
                }
            }
        }
    }
```

```
        for (const string& word : visitedNextLevel) {
            wordSet.erase(word);
        }
        visitedLevel = visitedNextLevel;
    }
    if (found) {
        vector<string> path;
        Backtrack(beginWord, endWord, graph, path, result);
    }
    return result;
}

int main() {
    string beginWord, endWord;
    int n;
    cout << "Enter the Value of N :- ";
    cin >> n;
    cout << "Enter the Begin Word :- ";
    cin >> beginWord;
    cout << "Enter the End Word :- ";
    cin >> endWord;
    vector<string> wordList(n);
    cout << "Enter the Word List :- ";
    for (int i = 0; i < n; ++i) {
        cin >> wordList[i];
    }

    vector<vector<string>> result = FindLadders(beginWord, endWord,
wordList);
    cout << "\nThe Shortest Transformation Sequences are :- " << endl <<endl;
    cout << "[";
    for (int i = 0; i < result.size(); ++i) {
```

```
        cout << "[";
        for (int j = 0; j < result[i].size(); ++j) {
            cout << "\"" << result[i][j] << "\"";
            if (j != result[i].size() - 1) {
                cout << ",";
            }
        }
        cout << "]";
        if (i != result.size() - 1) {
            cout << ",";
        }
    }
    cout << "]" << endl;
    return 0;
}
```

Output

```
Enter the Value of N :- 6
Enter the Begin Word :- hit
Enter the End Word :- cog
Enter the Word List :- hot dot dog lot log cog

The Shortest Transformation Sequences are :-

[["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]]
```