Experiment-3.2

Student Name: Mukund Jaiswal **UID: 21BCS3407**
Branch: CSE **Section/Group: 602-B**
Semester: 5th **Date of Performance: 26/10/23**
Subject Name: AP-I **Subject Code: 21CSP-314**

Aim- Backtracking: Implement the problems based on Backtracking.

Objectives- The objective of this experiment is to understand the concept of backtracking.

Problem1: https://www.hackerrank.com/challenges/crossword-puzzle/problem?isFullScreen=true&h_l=interview&playlist_slugs%5B%5D=interview-preparation-kit&playlist_slugs%5B%5D=recursion-backtracking

Problem2:https://www.hackerrank.com/challenges/ctci-recursive-staircase/problem?h_l=interview&isFullScreen=false&playlist_slugs%5B%5D=interview-preparation-kit&playlist_slugs%5B%5D=recursion-backtracking

Description-

Backtracking is a problem-solving algorithmic technique that involves finding a solution incrementally by trying **different options** and **undoing** them if they lead to a **dead end**. It is commonly used in situations where you need to explore multiple possibilities to solve a problem, like searching for a path in a maze or solving puzzles like Sudoku. When a dead end is reached, the algorithm backtracks to the previous decision point and explores a different path until a solution is found or all possibilities have been exhausted.

Types of Backtracking Problems

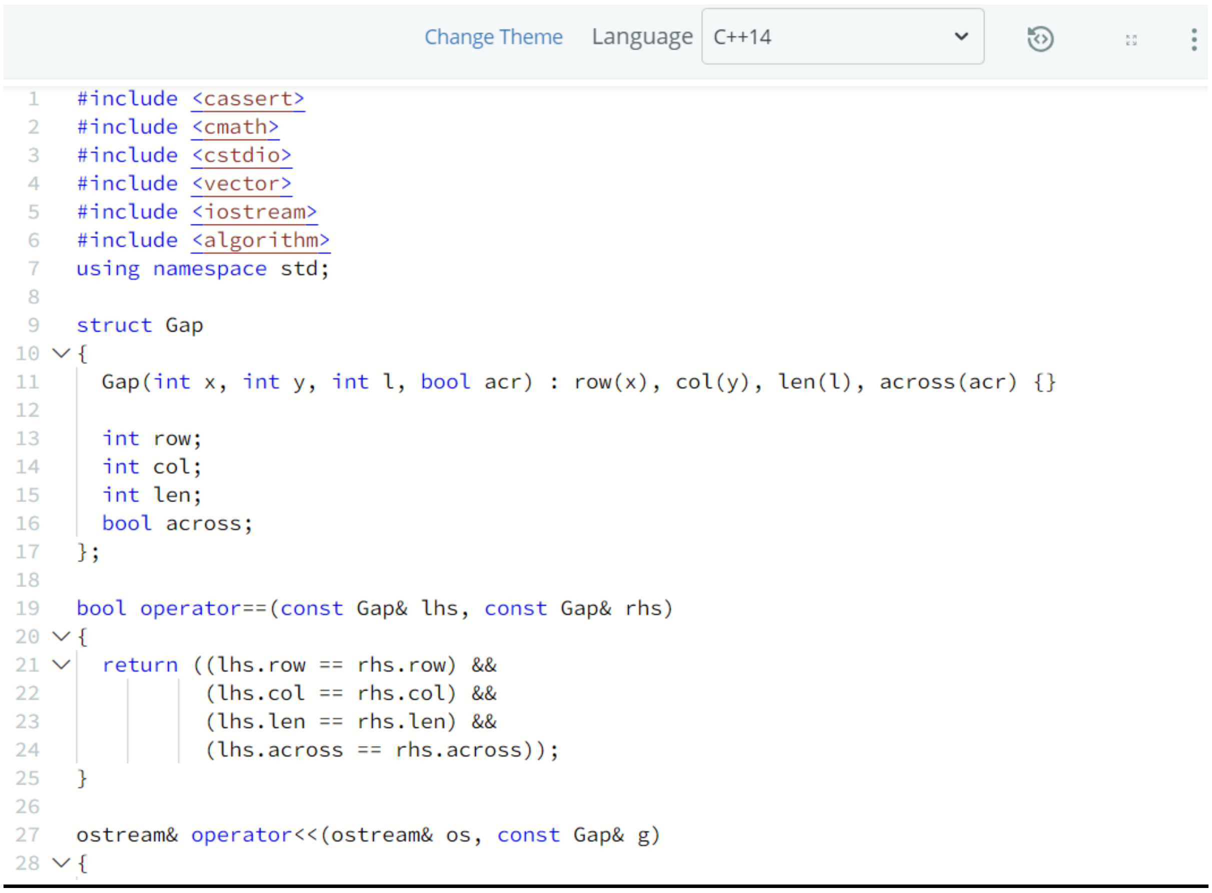
Problems associated with backtracking can be categorized into 3 categories:

- **Decision Problems:** Here, we search for a feasible solution.

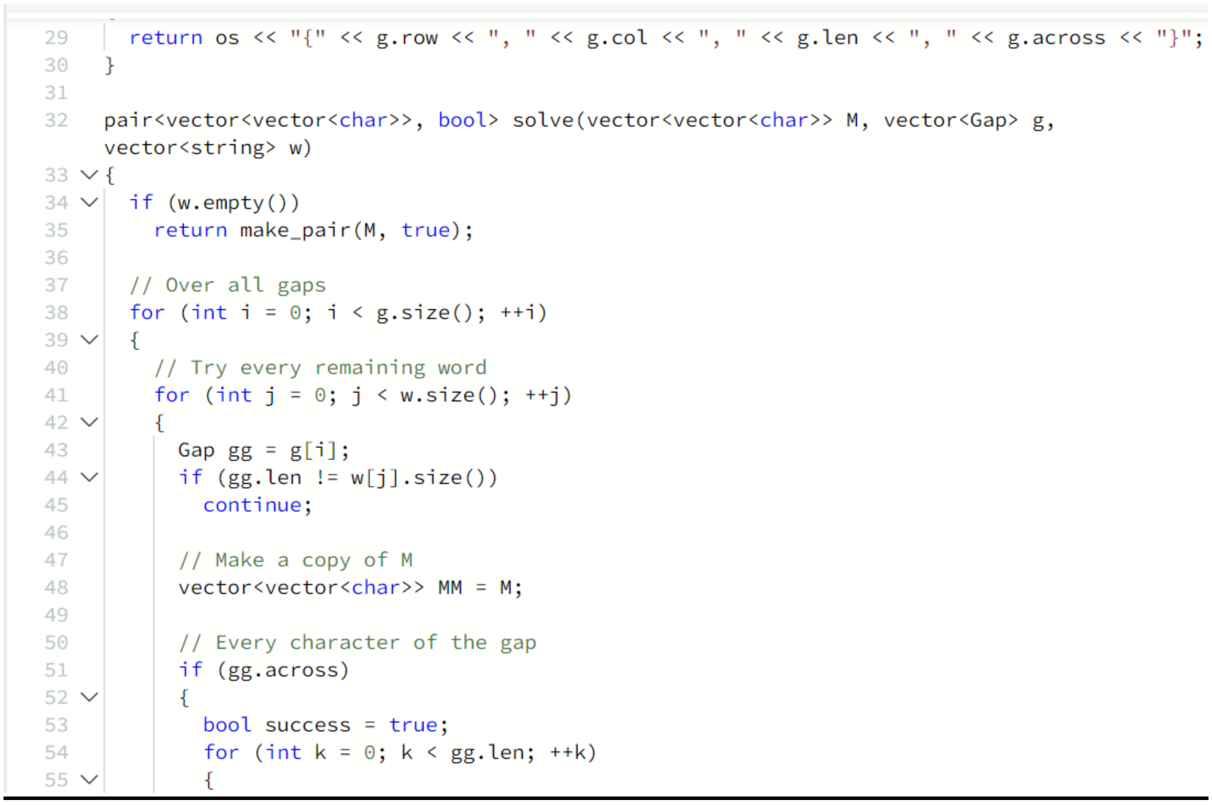
- **Optimization Problems:** For this type, we search for the best solution.
- **Enumeration Problems:** We find set of all possible feasible solutions to the problems of this type.

Code:

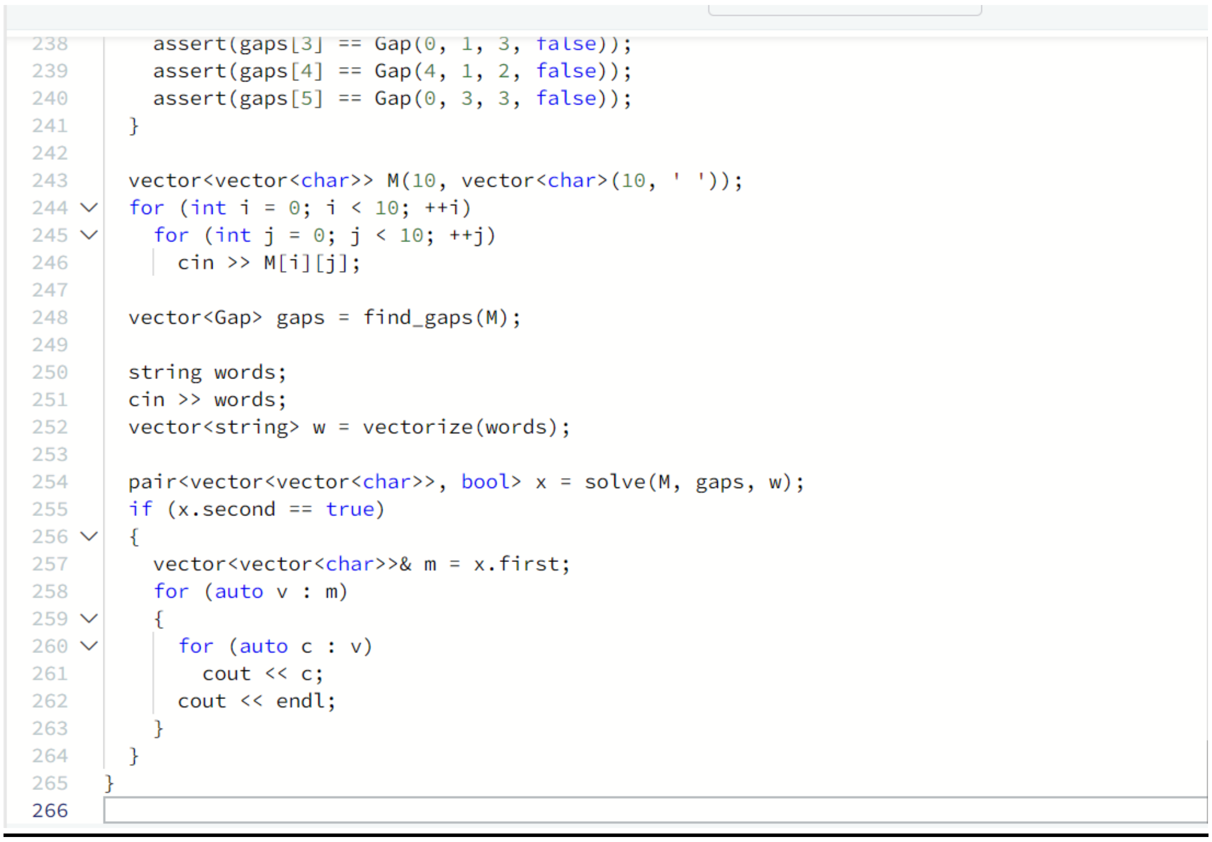
1.



```cpp
#include <cassert>
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

struct Gap
{
  Gap(int x, int y, int l, bool acr) : row(x), col(y), len(l), across(acr) {}

  int row;
  int col;
  int len;
  bool across;
};

bool operator==(const Gap& lhs, const Gap& rhs)
{
  return ((lhs.row == rhs.row) &&
          (lhs.col == rhs.col) &&
          (lhs.len == rhs.len) &&
          (lhs.across == rhs.across));
}

ostream& operator<<(ostream& os, const Gap& g)
{
```

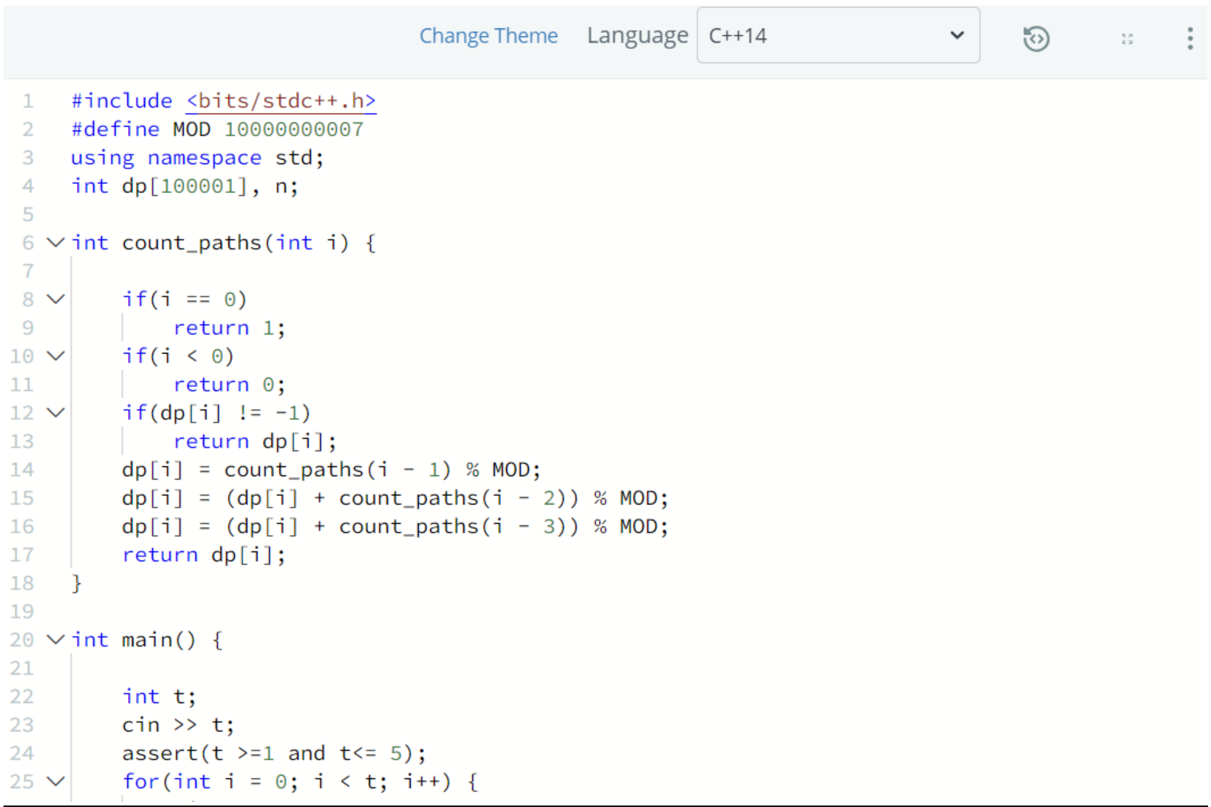


```cpp
    return os << "{" << g.row << ", " << g.col << ", " << g.len << ", " << g.across << "}";
}

pair<vector<vector<char>>, bool> solve(vector<vector<char>> M, vector<Gap> g, vector<string> w)
{
    if (w.empty())
        return make_pair(M, true);

    // Over all gaps
    for (int i = 0; i < g.size(); ++i)
    {
        // Try every remaining word
        for (int j = 0; j < w.size(); ++j)
        {
            Gap gg = g[i];
            if (gg.len != w[j].size())
                continue;

            // Make a copy of M
            vector<vector<char>> MM = M;

            // Every character of the gap
            if (gg.across)
            {
                bool success = true;
                for (int k = 0; k < gg.len; ++k)
                {
```

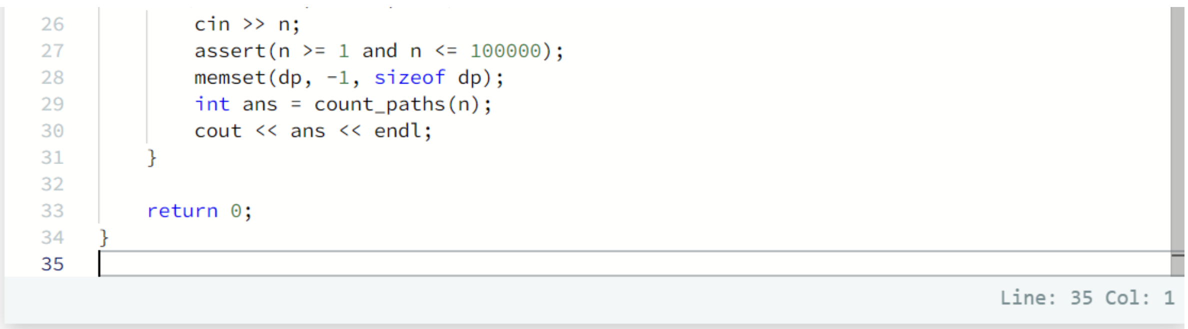


```cpp
        assert(gaps[3] == Gap(0, 1, 3, false));
        assert(gaps[4] == Gap(4, 1, 2, false));
        assert(gaps[5] == Gap(0, 3, 3, false));
    }

    vector<vector<char>> M(10, vector<char>(10, ' '));
    for (int i = 0; i < 10; ++i)
        for (int j = 0; j < 10; ++j)
            cin >> M[i][j];

    vector<Gap> gaps = find_gaps(M);

    string words;
    cin >> words;
    vector<string> w = vectorize(words);

    pair<vector<vector<char>>, bool> x = solve(M, gaps, w);
    if (x.second == true)
    {
        vector<vector<char>>& m = x.first;
        for (auto v : m)
        {
            for (auto c : v)
                cout << c;
            cout << endl;
        }
    }
}

```

2.



```cpp
#include <bits/stdc++.h>
#define MOD 10000000007
using namespace std;
int dp[100001], n;

int count_paths(int i) {

    if(i == 0)
        return 1;
    if(i < 0)
        return 0;
    if(dp[i] != -1)
        return dp[i];
    dp[i] = count_paths(i - 1) % MOD;
    dp[i] = (dp[i] + count_paths(i - 2)) % MOD;
    dp[i] = (dp[i] + count_paths(i - 3)) % MOD;
    return dp[i];
}

int main() {

    int t;
    cin >> t;
    assert(t >=1 and t<= 5);
    for(int i = 0; i < t; i++) {
```

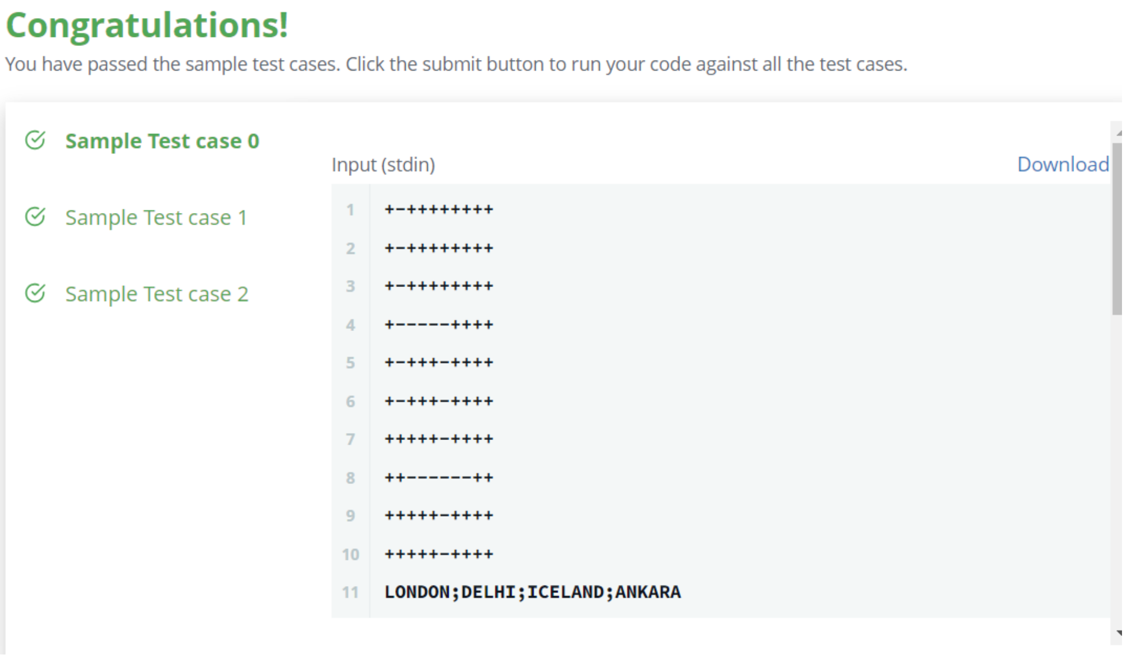


```cpp
        cin >> n;
        assert(n >= 1 and n <= 100000);
        memset(dp, -1, sizeof dp);
        int ans = count_paths(n);
        cout << ans << endl;
    }

    return 0;
}

```

Line: 35 Col: 1

Outcome-

Problem 1 outcome



Congratulations!

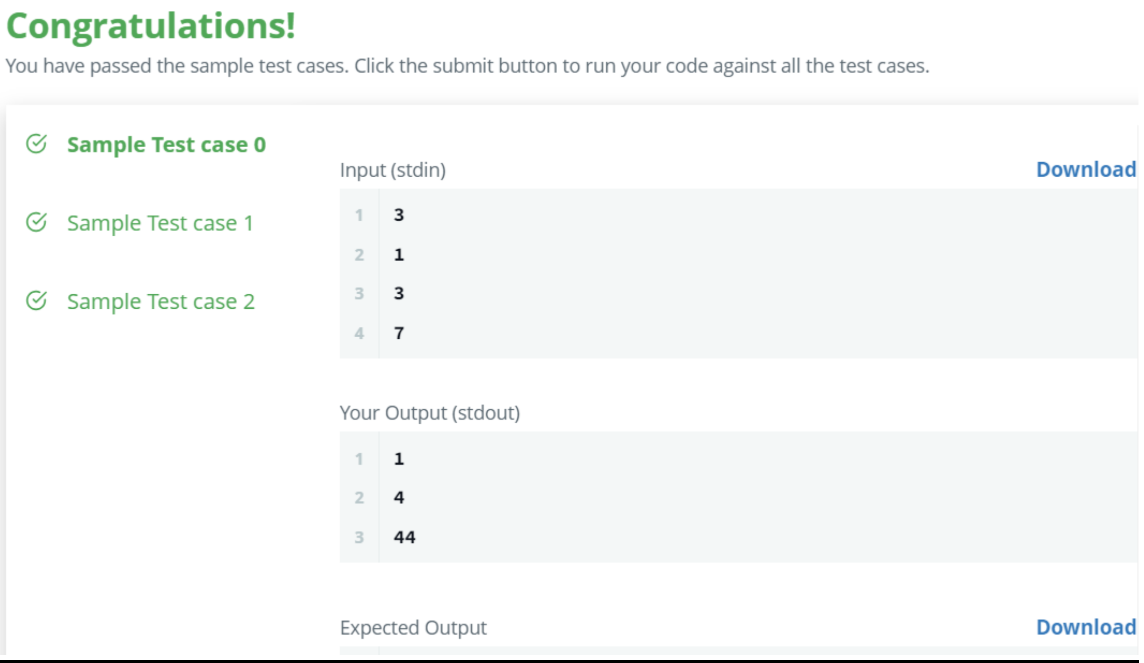
You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0
Sample Test case 1
Sample Test case 2

Input (stdin) Download

```
+-++++++++
+-++++++++
+-++++++++
+-----++++
+-+++-++++
+-+++-++++
+++++-++++
++------++
+++++-++++
+++++-++++
LONDON;DELHI;ICELAND;ANKARA
```

Problem 2 Outcome



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0
Sample Test case 1
Sample Test case 2

Input (stdin) Download

```
3
1
3
7
```

Your Output (stdout)

```
1
4
44
```

Expected Output Download

Learning Outcomes-

1. Learnt about the backtracking.
2. Learnt about staircase and crossword problems.