**Assignment 2**

**Problem Statement:**  
Implement a **Constraint Satisfaction Problem (CSP)** using the **Map Coloring Problem**.

**Theory**

**1. Constraint Satisfaction Problem (CSP)**

* A CSP is defined as a problem where we need to assign values to a set of variables subject to specific constraints.
* General representation:
  + **Variables (X):** Set of variables to be assigned.
  + **Domains (D):** Possible values each variable can take.
  + **Constraints (C):** Conditions that restrict which assignments are valid.

Examples of CSPs include Sudoku, N-Queens, Timetable scheduling, and Map coloring.

**2. Map Coloring Problem**

* **Objective:** Assign colors to regions of a map such that no two adjacent regions have the same color.
* **Variables:** The regions on the map (WA, NT, SA, Q, NSW, V, T).
* **Domains:** Colors {RED, GREEN, BLUE}.
* **Constraints:** Neighboring regions must not share the same color.

**3. CSP Solving Technique – Backtracking**

* Assign values to variables one by one.
* Check consistency after every assignment.
* If conflict arises, backtrack and try a different value.
* Continue until a complete solution is found or no solution exists.

**Algorithm Steps**

1. Start with all variables unassigned.
2. Select a variable that is not yet assigned.
3. Assign a color from the domain (RED, GREEN, BLUE).
4. Check if the assignment satisfies the constraints (neighbors have different colors).
5. If valid, continue assigning the next variable (recursive call).
6. If no valid assignment is possible, backtrack and try another color.
7. Stop when all variables are assigned.

**Code (C++ Implementation)**

#include <iostream>

#include <vector>

#include <map>

#include <string>

#include <algorithm>

using namespace std;

class MapColor {

public:

vector<string> REGIONS = {"WA", "NT", "SA", "Q", "NSW", "V", "T"};

vector<string> COLORS = {"RED", "GREEN", "BLUE"};

map<string, vector<string>> NEIGHBORS = {

{"WA", {"NT", "SA"}},

{"NT", {"WA", "SA", "Q"}},

{"SA", {"WA", "NT", "Q", "NSW", "V"}},

{"Q", {"NT", "SA", "NSW"}},

{"NSW", {"Q", "SA", "V"}},

{"V", {"SA", "NSW"}},

{"T", {}}

};

map<string, string> assignment;

bool backtrack() {

if (assignment.size() == REGIONS.size())

return true;

string var;

for (auto &r : REGIONS) {

if (assignment.find(r) == assignment.end()) {

var = r;

break;

}

}

for (auto &color : COLORS) {

if (isConsistent(var, color)) {

assignment[var] = color;

if (backtrack())

return true;

assignment.erase(var);

}

}

return false;

}

bool isConsistent(const string &var, const string &color) {

for (auto &nb : NEIGHBORS[var]) {

if (assignment.find(nb) != assignment.end() && assignment[nb] == color)

return false;

}

return true;

}

void printSolution() {

cout << "Solution:\n";

for (auto &r : REGIONS) {

cout << r << " = " << assignment[r];

if (!NEIGHBORS[r].empty()) {

cout << " | Neighbors: ";

for (auto &nb : NEIGHBORS[r]) {

cout << nb << "(";

if (assignment.find(nb) != assignment.end())

cout << assignment[nb];

else

cout << "Not Assigned";

cout << ") ";

}

}

cout << "\n";

}

}

};

int main() {

MapColor csp;

if (csp.backtrack()) {

csp.printSolution();

} else {

cout << "No solution found.\n";

}

return 0;

}

**Sample Output**

Solution:

WA = RED | Neighbors: NT(GREEN) SA(BLUE)

NT = GREEN | Neighbors: WA(RED) SA(BLUE) Q(RED)

SA = BLUE | Neighbors: WA(RED) NT(GREEN) Q(RED) NSW(GREEN) V(RED)

Q = RED | Neighbors: NT(GREEN) SA(BLUE) NSW(GREEN)

NSW = GREEN | Neighbors: Q(RED) SA(BLUE) V(RED)

V = RED | Neighbors: SA(BLUE) NSW(GREEN)

T = RED

**Conclusion**

* The **Map Coloring Problem** was successfully solved using a **Constraint Satisfaction Problem approach**.
* The **backtracking algorithm** was used to systematically assign colors to regions.
* The constraints ensured that no two neighboring regions shared the same color.
* This assignment demonstrates how CSPs can be applied to real-world problems like timetabling, scheduling, and resource allocation.