## Indian Institute of Information Technology Surat

****

# Lab Report on

# Artificial Intelligence (CS 701) Practical

**Submitted by**

### [RAHUL KUMAR SINGH] (UI21CS44)

**Course Faculty**

### Dr. Ritesh Kumar

### Mrs. Archana Balmik

## Department of Computer Science and Engineering

## Indian Institute of Information Technology Surat

## Gujarat-394190, India

**Aug-2024**

## Lab No: 9

## Aim:

Implement a versatile solution for cryptarithmetic puzzles that can handle multiple inputs.

## Description:

* Implement a backtracking algorithm for cryptarithmetic puzzles.
* Time complexity is ((n+1)×len(string)+10!+26).
* Assign unique digits to letters in equations.
* Verification of the correctness of digit assignments.
* Display results in a clear tabular format.
* Solution is capable of handling multiple inputs when adding the equation.

## Code:

**A)C++**

#include <bits/stdc++.h>

using namespace std;

mt19937\_64 gen(chrono::steady\_clock::now().time\_since\_epoch().count());

uniform\_int\_distribution<long long> rnd(0,LLONG\_MAX);

#define ll long long

const int MAXN = 3e6 + 5;

const int MAX\_N = 15;

const int MOD = 1e9 + 7;

const int INF = 1e9;

const ll LINF = 1e18;

int n, x[MAX\_N], y[MAX\_N], adj[MAX\_N][MAX\_N], dp[1 << MAX\_N][MAX\_N];

ll tsp(ll mask, ll u) {

if (mask == (1 << n) - 1) return adj[u][0];

if (dp[mask][u] != -1) return dp[mask][u];

ll ans = INF;

for (ll v = 0; v < n; v++) {

if (!(mask & (1 << v))) {

ll cur = adj[u][v] + tsp(mask | (1 << v), v);

ans = min(ans, cur);

}

}

return dp[mask][u] = ans;

}

ll hsh[26];

vector<int> uniq;

ll comb[26];

int used[10];

int charFront[26];

bool bts(int sum, int cur) {

if (cur == uniq.size()) {

return (sum == 0);

}

int ch = uniq[cur];

if (comb[ch] != -1) {

bool check = bts(sum + hsh[ch] \* comb[ch], cur + 1);

if (check) return true;

}

for (int i = 9; i >= 0; i--) {

if (charFront[ch] == 1 && i == 0) continue;

if (used[i]) continue;

used[i] = 1;

comb[ch] = i;

bool check = bts(sum + hsh[ch] \* i, cur + 1);

if (check) return true;

comb[ch] = -1;

used[i] = 0;

}

return false;

}

void display\_table(const vector<string>& st) {

cout << "\nMapped Values:\n";

cout << "Character | Value\n";

cout << "------------------\n";

for (int i = 0; i < 26; i++) {

if (comb[i] != -1) {

char ct = 'A' + i;

cout << " " << ct << " | " << comb[i] << endl;

}

}

cout << "\nEquations:\n";

for (const auto& s : st) {

for (char c : s) {

cout << comb[c - 'A'] << " ";

}

cout << endl;

}

}

void sol() {

ll a, b, c, n, m, k = -1, x, resu = 0;

cin >> n;

vector<string> st(n + 1);

for (int i = 0; i <= n; i++) {

cin >> st[i];

}

int ch = 0;

ll exp = 0;

uniq.clear();

for (int i = 0; i < 26; i++) comb[i] = -1;

for (int i = 0; i < n; i++) {

exp = 1;

for (int j = st[i].size() - 1; j >= 0; j--) {

ch = st[i][j] - 'A';

if (hsh[ch] == 0) uniq.push\_back(ch);

hsh[ch] += exp;

exp \*= 10;

if (j == 0) charFront[ch] = 1;

}

}

exp = 1;

int i = n;

for (int j = st[i].size() - 1; j >= 0; j--) {

ch = st[i][j] - 'A';

if (hsh[ch] == 0) uniq.push\_back(ch);

hsh[ch] -= exp;

exp \*= 10;

if (j == 0) charFront[ch] = 1;

}

if (bts(0, 0)) {

cout << "There exists a valid combination!\n";

display\_table(st);

} else {

cout << "No solution exists!\n";

}

}

int main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0); cout.tie(0);

int tc = 1;

for (int t = 1; t <= tc; t++) {

sol();

}

return 0;

}

**B)Python**

**from itertools import permutations**

**from tabulate import tabulate**

**def disResult(equation):**

**left, right = equation.split('=')**

**left\_words = left.split('+')**

**return [word.strip() for word in left\_words], right.strip()**

**def isValid(mapping, left\_words, right\_word):**

**left\_sum = sum(int(''.join(str(mapping[ch]) for ch in word)) for word in left\_words)**

**right\_value = int(''.join(str(mapping[ch]) for ch in right\_word))**

**return left\_sum == right\_value**

**def solveCrypt(equation):**

**left\_words, right\_word = disResult(equation)**

**unique\_chars = set(''.join(left\_words) + right\_word)**

**if len(unique\_chars) > 10:**

**return None**

**for perm in permutations(range(10), len(unique\_chars)):**

**mapping = dict(zip(unique\_chars, perm))**

**if any(mapping[word[0]] == 0 for word in left\_words + [right\_word]):**

**continue**

**if isValid(mapping, left\_words, right\_word):**

**return mapping**

**return None**

**def displayResult(equation, mapping):**

**left\_words, right\_word = disResult(equation)**

**mapped\_left\_words = [''.join(str(mapping[ch]) for ch in word) for word in left\_words]**

**mapped\_right\_value = ''.join(str(mapping[ch]) for ch in right\_word)**

**data = [**

**["Input Equation", equation],**

**["Mapped Values", " + ".join(mapped\_left\_words) + " = " + mapped\_right\_value],**

**["Result", "Valid Solution"]**

**]**

**print(tabulate(data, headers="firstrow", tablefmt="grid"))**

**def main():**

**equation = input("Enter the cryptarithmetic equation (SEND + MORE = MONEY): ")**

**solution = solveCrypt(equation)**

**if solution:**

**print("Solution found:")**

**for char, digit in solution.items():**

**print(f"{char} -> {digit}")**

**displayResult(equation, solution)**

**else:**

**print("No solution exists.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

## Output:

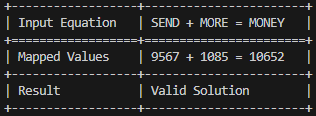
**A)C++**

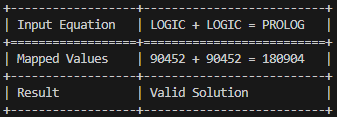
****

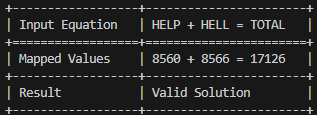
****

****

**B)Python**

****

****

****

## Conclusion:

* The backtracking algorithm effectively solves cryptarithmetic puzzles with unique digit assignments.
* Results are clearly displayed in a structured tabular format for easy interpretation.
* The solution efficiently handles multiple equations and character mappings simultaneously.
* Overall, the implementation demonstrates a robust & efficient approach to combinatorial problem-solving.