Source Code

```
//  CS 4613
//  Project 2
//
//  Micheal Zhang

/*
 * Variables:
 *   x9: 1
 *   c3, c2, c1: {0, 1}
 *   x1, x5: {1, 2, 3, 4 ,5, 6, 7, 8, 9}
 *   else: {0, 1, 2, 3, 4 ,5, 6, 7, 8, 9}
 *
 * Constraints:
 *   x4 + x8 = 10 * c1 + x13
 *   c1 + x3 + x7 = 10 * c2 + x12
 *   c2 + x2 + x6 = 10 * c3 + x11
 *   c3 + x1 + x5 = 10 * x9 + x10
 *
 * Initial Degrees:
 *   x2, x3, x6, x7, x9, x11, x12: 4
 *   x1, x4, x5, x8, x10, x13: 3
 *   c1, c3: 7
 *   c2: 8
 */

#define x1 0
#define x2 1
#define x3 2
#define x4 3
#define x5 4
#define x6 5
#define x7 6
#define x8 7
#define x9 8
#define x10 9
#define x11 10
#define x12 11
#define x13 12
#define c1 13
#define c2 14
#define c3 15

#include <iostream>
```

```cpp
#include <fstream>
#include <algorithm>
#include <queue>

using namespace std;

class CSP {
private:
    int assignment[16];
    vector<int> remaining_values[16];
    int degree[16];  /* Number of constraints on other unassigned variables */
    bool same_letter[13][13];  /* Letter constraints */
    vector<pair<int, int> > constraints;

public:
    CSP() : degree{3, 4, 4, 3, 3, 4, 4, 3, 4, 3, 4, 4, 3, 7, 8, 7} {
        for (int i = 0; i < 16; i++) {
            assignment[i] = -1;  /* Dummy value for unassigned variables */
            if (i >= c1)
                for (int j = 1; j >= 0; j--)  /* 0 ~ 1 */
                    remaining_values[i].push_back(j);
            else if (i == x1 || i == x5)
                for (int j = 9; j >= 1; j--)  /* 1 ~ 9 */
                    remaining_values[i].push_back(j);
            else if (i == x9)
                continue;
            else
                for (int j = 9; j >= 0; j--)  /* 0 ~ 9 */
                    remaining_values[i].push_back(j);
        }
        for (int i = 0; i < 13; i++)  /* Default all letters are different */
            for (int j = 0; j < 13; j++)
                same_letter[i][j] = false;
        assignment[x9] = 1;
    }

    CSP(const CSP& other) {
        for (int i = 0; i < 16; i++) {
            assignment[i] = other.assignment[i];
            degree[i] = other.degree[i];
            remaining_values[i] = other.remaining_values[i];
        }
        for (int i = 0; i < 13; i++)
            for (int j = 0; j < 13; j++)
                same_letter[i][j] = other.same_letter[i][j];
    }
```

```
    bool is_complete() {
        for (int i = 0; i < 16; i++)
            if (assignment[i] == -1)
                return false;
        return true;
    }

    bool is_consistent() {
        if (assignment[x4] != -1 && assignment[x8] != -1 && assignment[c1] != -1 &&
assignment[x13] != -1)
            if (assignment[x4] + assignment[x8] != 10 * assignment[c1] + assignment[x13])
                return false;
        if (assignment[c1] != -1 && assignment[x3] != -1 && assignment[x7] != -1 && assignment[c2]
!= -1 && assignment[x12] != -1)
            if (assignment[c1] + assignment[x3] + assignment[x7] != 10 * assignment[c2] +
assignment[x12])
                return false;
        if (assignment[c2] != -1 && assignment[x2] != -1 && assignment[x6] != -1 && assignment[c3]
!= -1 && assignment[x11] != -1)
            if (assignment[c2] + assignment[x2] + assignment[x6] != 10 * assignment[c3] +
assignment[x11])
                return false;
        if (assignment[c3] != -1 && assignment[x1] != -1 && assignment[x5] != -1 &&
assignment[x10] != -1)
            if (assignment[c3] + assignment[x1] + assignment[x5] != 10 * assignment[x9] +
assignment[x10])
                return false;
        for (pair<int, int> &i : constraints)
            if (assignment[i.first] != -1 && assignment[i.second] != -1)
                if (assignment[i.first] != assignment[i.second])
                    return false;
        for (int i = 0; i < 13; i++)
            for (int j = i + 1; j < 13; j++)
                if (assignment[i] != -1 && assignment[j] != -1)
                    if (same_letter[i][j] != (assignment[i] == assignment[j]))
                        return false;
        return true;
    }

    void append_constraint(int i, int j) {
        same_letter[i][j] = true;
        same_letter[j][i] = true;
        degree[i]++;
        degree[j]++;
    }
```

```
void assign(int var, int val) {
    assignment[var] = val;
    vector<int> relation[16] {
        {x5, x10, c3},      // x1
        {x6, x11, c2, c3},  // x2
        {x7, x12, c1, c2},  // x3
        {x8, x13, c1},      // x4
        {x1, x10, c3},      // x5
        {x2, x11, c2, c3},  // x6
        {x3, x12, c1, c2},  // x7
        {x4, x13, c1},      // x8
        {},                 // x9
        {x1, x5, c3},       // x10
        {x2, x6, c2, c3},   // x11
        {x3, x7, c1, c2},   // x12
        {x4, x8, c1},       // x13
        {x3, x4, x7, x8, x12, x13, c2},      // c1
        {x2, x3, x6, x7, x11, x12, c1, c3},  // c2
        {x1, x2, x5, x6, x10, x11, c2}       // c3
    };
    for (int i : relation[var])
        degree[i]--;
}

int select_unassigned_variable() {
    vector<int> selected;
    int min = 10;
    for (int i = 0; i < 16; i++) {  /* Minimum remaining values */
        if (assignment[i] != -1)
            continue;
        if (remaining_values[i].size() < min) {
            min = remaining_values[i].size();
            selected.clear();
            selected.push_back(i);
        }
        else if (remaining_values[i].size() == min)
            selected.push_back(i);
    }
    if (selected.size() == 1)
        return selected[0];
    int max = -1, final_selected;
    for (int i : selected)  /* Degree heurisitics */
        if (degree[i] > max) {
            max = degree[i];
            final_selected = i;
```

```
            }
        return final_selected;
    }

    int order_domain_values(int var) {
        int val = -1;
        if (remaining_values[var].size() > 0)
            val = remaining_values[var].back();
            remaining_values[var].pop_back();
        return val;
    }

    friend ostream& operator<<(ostream& os, const CSP &csp) {
        os << csp.assignment[x1] << csp.assignment[x2] << csp.assignment[x3] << csp.assignment[x4]
<< endl;
        os << csp.assignment[x5] << csp.assignment[x6] << csp.assignment[x7] << csp.assignment[x8]
<< endl;
        os << csp.assignment[x9] << csp.assignment[x10] << csp.assignment[x11] <<
csp.assignment[x12] << csp.assignment[x13] << endl;
        return os;
    }
};

bool backtrack(ostream& os, CSP &csp) {
    if (csp.is_complete()) {
        os << csp;
        // cout << csp;
        return true;
    }
    int var = csp.select_unassigned_variable();
    for (int val = csp.order_domain_values(var); val != -1; val = csp.order_domain_values(var)) {
        CSP new_csp(csp);
        new_csp.assign(var, val);
        if (new_csp.is_consistent())
            if (backtrack(os, new_csp))
                return true;
    }
    return false;
}

int main(int argc, const char *argv[]) {
    ifstream fin;
    fin.open("input.txt");
    if (!fin.good())
        cerr << "Failed to open input.txt" << endl;
    string input;
```

```
    for (int i = 0; i < 3; i++) {
        string line;
        getline(fin, line);
        input += line;
    }
    fin.close();
    if (input.length() != 13)
        cerr << "Bad input" << endl;
    CSP csp;
    for (int i = 0; i < input.length(); i++)
        for (int j = i + 1; j < input.length(); j++)
            if (input[i] == input[j]) {
                // cout << "new rule: x" << i + 1 << " == x" << j + 1 << endl;
                csp.append_constraint(i, j);
            }
    ofstream fout;
    fout.open("output.txt");
    if (!fout.good())
        cerr << "Failed to open output.txt" << endl;
    if (!backtrack(fout, csp))
        cout << "Failure" << endl;
    fout.close();
    return 0;
}
```

Output 1

```
9567
1085
10652
```

Output 2

```
7483
7455
14938
```