Homework 1: Solving Sudoku by SAT

> Practice/Real-Life Applications of Computational Algorithms, Spring 2021

Run

1. `g++ -std=c++17 -o solver solver.cpp`
2. `./solver [Input Filename] [Output Filename] ./MiniSat_v1.14_linux`

Input Spec

1. has a size NxN, and
2. is prefilled with numbers 0 to N, where 0 represents the square is empty.

```
0 6 0 1 0 4 0 5 0
0 0 8 3 0 5 6 0 0
2 0 0 0 0 0 0 0 1
8 0 0 4 0 7 0 0 6
0 0 6 0 0 0 3 0 0
7 0 0 9 0 1 0 0 4
5 0 0 0 0 0 0 0 2
0 0 7 2 0 6 9 0 0
0 4 0 5 0 8 0 7 0
```

Output Spec

1. A NxN filled puzzle.
2. If the input is not solvable, then the output will be "NO".

Implementation

1. Read input file.
2. Generate the CNF witch let each cell, row, column, and block satisfied the rule
3. Add the prefilled cell to the CNF
4. Use `system()` to call MiniSAT to get the satisfiability and a solution.
5. Output the solution to the output file.

CNF

Define the variable as $V_{row,col,num}$, which means weather the cell at (row, col) is num.

- Each cell chould be assigned as exactly one number, so the clause $\vee_{k\in[N]}V_{row,col,k}$ and $-V_{row,col,i} \vee -V_{row,col,j}, \forall i \neq j \in [N]$ should be included in the CNF for each cell (row, col).
- Each number chould be assigned exactly once in each col, so the clause $\vee_{k\in[N]}V_{k,col,num}$ and $-V_{i,col,num} \vee -V_{j,col,num}, \forall i \neq j \in [N]$ should be included in the CNF for each column col and each number num.
- Each number chould be assigned exactly once in each row, so the clause $\vee_{k\in[N]}V_{row,k,num}$ and $-V_{row,i,num} \vee -V_{row,j,num}, \forall i \neq j \in [N]$ should be included in the CNF for each row row and each number num.
- Each number chould be assigned exactly once in each $\sqrt{N} \times \sqrt{N}$ block, so the clause $\vee_{(i,j)\in[a,b]\times[c,d]}V_{i,j,num}$ and $-V_{i,j,num} \vee -V_{x,y,num}, \forall (i,j), (x,y) \in [a,b] \times [c,d]$ should be included in the CNF for each block $[a,b] \times [c,d]$ and each number num.

Code

```cpp
class Puzzle {
private:
    vector<vector<int>> cell;

    // hash the variable to a id
    int id(int x, int y, int z);

    // decode the id
    tuple<int, int, int> reId(int id);

    // change the variables
    // witch exacyly one of them
    // should be assigned true to the clauses
    void genClause(vector<vector<int>> &CNF,
                   const vector<int> &v);

    // generate the CNF of basic rules
    void genCNF(vector<vector<int>> &CNF);

    // write the CNF to the output file,
    // call the MiniSAT,
    // and store the result into tmp file
    void callMiniSAT(const char *tmpFile,
                     const char *outFile,
                     const char *MiniSAT,
                     const vector<vector<int>> &CNF);

    // read the result of assigned variable from tmp file,
    // decode the variables to the cells,
    // and then write the output file
    void output(const char *tmpFile,
                const char *outFile);
public:
    Puzzle() {}

    // read the input file
    Puzzle(const char* filename);

    void solve(const char *outFile, const char *MiniSAT);
};

int main(int argc, char *argv[]) {
    Puzzle P(argv[1]);
    P.solve(argv[2], argv[3]);
    return 0;
}
```