CS-UY 4613 Project 2

Charles Pan, cp3723

Nick Li, ql2015

Instructions

Uncompressed the .zip file. The project has been compiled with Visual Studio 2022 and C++ 14. To run the program, run the Futoshiki.exe, the output file will be in the same directory, named from Output1.txt to Output3.txt. The source code is in the directory named Futoshiki.

Source Code

main.cpp

```
// author
// Charles Pan, cp3723
// Nick Li, ql2015

#include <string>
#include <vector>
#include <iostream>
#include <fstream>
#include <set>
#include <utility>
#include "Futoshiki.h"


int main() {
  Board board;
  for (int i = 1; i <= 3; ++i) {
    std::cout << "input " << i << std::endl;
    std::ifstream ifs("Input" + std::to_string(i) + ".txt");
    ifs >> board; // read input file
    ifs.close();
    board.solve();
    std::cout << "----------------------\n";
    std::cout << board;
    std::ofstream ofs;
    ofs.open("Output" + std::to_string(i) + ".txt");
    ofs << board; // write to the output file
    ofs.close();
```

```
    std::cout << "----------------------\n\n";
    board.clear();
  }
}
```

Futoshiki.h

```
#ifndef FUTOSHIKI_H
#define FUTOSHIKI_H

// author
// Charles Pan, cp3723
// Nick Li, ql2015

#include <string>
#include <vector>
#include <iostream>
#include <fstream>
#include <set>
#include <utility>


enum direction;


struct Cell
{
public:
  int value;
  int degree;

  Cell();
  Cell(int v, int degree);
  Cell(const Cell&);
  std::set<int> domain;
  std::vector<std::pair<direction, std::string>> constraints;

  Cell& operator=(const Cell& rhs);
};



class Board {
  friend std::ostream& operator<<(std::ostream& os, const Board& rhs);
  friend std::istream& operator>>(std::istream& is, Board& rhs);
public:
  int filled;
  std::vector<Cell> cells;


  Board();
  Board(const Board&);
  // decrease domain for a whole line
```

```
  void updateDomainCross(int x, int y, int v);
  int selectUnassignedVariable();
  void update_domain_by_constraits();
  bool checkComplete() { return filled == 25; };
  bool solve();
  void clear() { filled = 0; cells.clear(); }

  Board& operator=(const Board& rhs);
};



#endif // !FUTOSHIKI_H
```

Futoshiki.cpp

```
// author
// Charles Pan, cp3723
// Nick Li, ql2015

#include <string>
#include <vector>
#include <iostream>
#include <fstream>
#include <set>
#include <utility>
#include "Futoshiki.h"

using namespace std;

enum direction
{
  u = 0, // up
  d = 1, // down
  l = 2, // left
  r = 3  // right
};

Cell::Cell() {
  value = -1;
  degree = -1;
}

Cell::Cell(int v, int degree = 8) {
  this->value = v;
  this->degree = degree;
  for (int i = 1; i <= 5; i++) this->domain.insert(i);
}

Cell::Cell(const Cell& rhs) {
  this->value = rhs.value;
  this->domain = rhs.domain;
  this->degree = degree;
```

```
  this->constraints = rhs.constraints;
}

Cell& Cell::operator=(const Cell& rhs) {
  this->value = rhs.value;
  this->domain = rhs.domain;
  this->degree = rhs.degree;
  return *this;
}

////////////////////////////// BOARD ////////////////////////////////

// overload = opeartor
Board& Board::operator=(const Board& rhs) {
  this->filled = rhs.filled;
  this->cells = rhs.cells;
  return *this;
}

// output solution
ostream& operator<<(ostream& os, const Board& rhs) {
  int i, j;
  for (j = 0; j < 5; j++) {
    for (i = 0; i < 5; i++) {
      os << rhs.cells[i + 5 * j].value;
      if (i != 4) os << " ";
    }
    os << endl;
  }
  return os;
}
// input file data
istream& operator>>(istream& is, Board& rhs) {
  // initialize
  for (int i = 0; i < 25; ++i) {
    Cell c(0, 8);
    rhs.cells.push_back(c);
  }

  // fill in initial number
  int value;
  for (int j = 0; j < 5; j++) {
    for (int i = 0; i < 5; i++) {
      is >> value;
      // not blank
      if (value != 0) {
        rhs.cells[j * 5 + i].value = value;
        rhs.updateDomainCross(i, j, value);
        rhs.filled++;
      }
    }
  }

  // read constraints
  string symbol;
  // horizontal constraints
  // 5 rows of 4 constraints
  for (int j = 0; j < 5; j++) {
```

```
    for (int i = 0; i < 4; i++) {
      is >> symbol;
      if (symbol != "0") {
        rhs.cells[j * 5 + i].constraints.push_back({ r, symbol });
        rhs.cells[j * 5 + i + 1].constraints.push_back({ l, symbol });
      }
    }
  }
  // vertical constraints
  // 4 rows of 5 constraints
  for (int j = 0; j < 4; j++) {
    for (int i = 0; i < 5; i++) {
      is >> symbol;
      if (symbol != "0") {
        rhs.cells[j * 5 + i].constraints.push_back({ d, symbol });
        rhs.cells[(j + 1) * 5 + i].constraints.push_back({ u, symbol });
      }
    }
  }

  // update domains according to the constraits
  rhs.update_domain_by_constraits();

  return is;
}

Board::Board() {
  filled = 0;
}

// copy constructor
Board::Board(const Board& rhs) {
  this->cells = rhs.cells;
  this->filled = rhs.filled;
}

// clear domain of current element
// update domains, domain_size, degree of elements on the same row and column
void Board::updateDomainCross(int x, int y, int v) {
  cells[y * 5 + x].domain.clear();
  // vertical
  for (int i = x; i < 25; i += 5) {
    if (cells[i].domain.erase(v)) {
      cells[i].degree--;
    }
  }
  // horizontal
  for (int i = y * 5; i < (y * 5 + 5); i++) {
    if (cells[i].domain.erase(v)) {
      cells[i].degree--;
    }
  }
}

// first apply MRV
// then degree heuristic
// return the index of the selected element
int Board::selectUnassignedVariable() {
```

```
  int domain_size_min = 6;
  int degree_max = -1;
  int index = -1;
  for (int i = 0; i < 25; ++i) {
    if (cells[i].value != 0) continue; // skip assigned elements
    if (cells[i].domain.size() < domain_size_min) {
      // MRV
      domain_size_min = cells[i].domain.size();
      degree_max = cells[i].degree;
      index = i;
    }
    else if (cells[i].domain.size() == domain_size_min) {
      // compare degree
      if (cells[i].degree > degree_max) {
        degree_max = cells[i].degree;
        index = i;
      }
    }
  }
  return index;
}

// helper function
// sign == 0, cell.v < value; sign == 1, cell.v > value;
inline void update_cell(std::set<int>& domain, int value, int sign) {
  if (sign == 0) {
    for (int i = value; i <= 5; ++i) {
      domain.erase(i);
    }
  }
  else {
    for (int i = value; i >= 0; --i) {
      domain.erase(i);
    }
  }
}

void Board::update_domain_by_constraits() {
  for (int i = 0; i < 25; ++i) {
    if (cells[i].value != 0) {
      // update domain of adjacent elements
      for (int j = 0; j < cells[i].constraints.size(); ++j) {
        direction dir = cells[i].constraints[j].first;
        string s = cells[i].constraints[j].second;
        if (dir == u) {
          update_cell(cells[i - 5].domain, cells[i].value, s == "^" ? 0 : 1);
        }
        else if (dir == d) {
          update_cell(cells[i + 5].domain, cells[i].value, s == "v" ? 0 : 1);
        }
        else if (dir == l) {
          update_cell(cells[i - 1].domain, cells[i].value, s == "<" ? 0 : 1);
        }
        else { // dir == r
          update_cell(cells[i + 1].domain, cells[i].value, s == ">" ? 0 : 1);
        }
      }
    }
```

```
  }
}

bool Board::solve() {
  if (checkComplete()) {
    return true;
  }
  int index = selectUnassignedVariable(); // find next element
  // if no available candidates, return false
  if (cells[index].domain.size() == 0) return false;
  for (int i = 1; i <= 5; ++i) {
    if (cells[index].domain.count(i) == 0) continue;
    Board new_board(*this);
    new_board.cells[index].value = i; // fill in the value
    new_board.filled++;
    new_board.updateDomainCross(index % 5, index / 5, i);
    new_board.update_domain_by_constraits();
    if (new_board.solve()) { // if find the answer, return directly
      *this = new_board;
      return true;
    }
  }
  return false;
}
```

Output

Output1.txt

3 2 4 1 5
5 3 1 2 4
1 5 2 4 3
2 4 5 3 1
4 1 3 5 2

Output2.txt

1 5 3 2 4
3 2 1 4 5
5 4 2 1 3
4 1 5 3 2
2 3 4 5 1

Output3.txt

3 4 2 5 1
1 5 4 2 3
2 3 5 1 4
5 1 3 4 2
4 2 1 3 5