МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра Информатики и систем управления

Лабораторная работа №4 «Рекурсия и головоломки.»

ОТЧЕТ по лабораторной работе № 4

по дисциплине

Технологии программирования

РУКОВОДИТЕЛЬ:

_____ Капранов С.Н.
(подпись) (фамилия, и.,о.)

СТУДЕНТ:

_____ Куликова Е.А.
(подпись) (фамилия, и.,о.)

18-ИСТ-4
(шифр группы)

Работа защищена «___» _____

С оценкой _____

Нижний Новгород

2020

Содержание

Введение

Первые головоломки появились в четвертом тысячелетии до нашей эры, однако уровень, которого достигло человечество в наше время, в праве даёт людям нынешнего поколения возможность называться победителями в деле создания бесполезных задачек.

В рамках четвёртой лабораторной работы (вариант 23) необходимо реализовать одну из усложнённых форм всем известной головоломки, как «Судоку» – первые встречные («Easy as ABC», «ABC End View», «Last Man Standing»; еще одно название – Букварики) – логическая головоломка с буквами. Необходимо заполнить квадратную сетку латинскими буквами (например, от A до E), так чтобы каждый символ встречался в каждой строке и в каждом столбце ровно один раз. Некоторые клетки сетки могут быть пустыми. Буква, стоящая на границе сетки, показывает – какая буква встретится первой в данной строке (столбце).

1. Цель работы

Создать программу, соответствующую правилам головоломки «Первые встречные», которая должна решать стандартные игровые карты путём некого алгоритма, используемого над входящим файлом.

2. Задачи

Поставленные задачи:

1. Разработать алгоритм, по которому будет выполняться программа.
2. Написать код, реализующий задание.
3. Протестировать, чтобы убедиться в правильности решения.

3. Описание алгоритма

Вход: Размер поля, само поле головоломки Первые встречные, часть клеток которого может быть заполнена, и боковые значения поля;

Выход: Полностью заполненное поле или пустое поле, если решения нет;

Начало

Пока возможно:

Для каждой клетки проверяем выполнение условий на уникальность в ряду и столбце:

Если для какой-то клетки подходящей цифры не нашлось, то завершаем работу алгоритма (решения нет);

Если существует единственная подходящая цифра, то заполняем клетку соответствующим образом;

Если все клетки заполнены, то завершаем цикл и возвращаем найденное решение;

Иначе если ни одну клетку за проход заполнить не удалось, то завершаем цикл;

Для клетки с минимальным количеством вариантов:

Пробуем ставить каждую букву по порядку и рекурсивно решать получившиеся Первые встречные;

Если решение было найдено, то возвращаем его;

Конец.

4. Код программы

```
#include <iostream>
#include <fstream>
#include <vector>
#include <set>

typedef std::vector<std::vector<int>> Puzzle;
typedef std::set<int> Values;

int size;

// Класс решатель головоломки Easy as ABC
class ABCSolver
{
public:
	// Поиск решения
	static Puzzle solve(const Puzzle& puzzle, const Puzzle& info)
	{
		Puzzle solution = puzzle;
		if (solveHelper(&solution, info))
			return solution;
		return Puzzle();
	}

	// Вспомогательная функция поиска решения
	static bool solveHelper(Puzzle* solution, const Puzzle& info)
	{
		int minRow = -1;
		int minColumn = -1;
		Values minValues;
		while (true)
		{// Пока возможно
			minRow = -1;
			for (int i = 0; i < size; i++)
			{
				for (int j = 0; j < size; j++)
				{
					if ((*solution)[i][j] != 0)
						continue;
					// Для каждой клетки ищем возможные значения
					Values possibleValues = findPossibleValues(i, j,
					*solution, info);
					int possibleVaueCount = possibleValues.size();
					// Если для какой-то клетки подходящего значения не
					   нашлось, то завершаем работу алгоритма (решения нет)
					if (possibleVaueCount == 0)
						return false;
					// Если существует единственное подходящее значение, то
					   заполняем клетку соответствующим образом
```

```
                                if (possibleVaueCount == 1)
                                        (*solution)[i][j] = *possibleValues.begin();
                                if (minRow < 0 || possibleVaueCount < minValues.size())
                                {
                                        minRow = i;
                                        minColumn = j;
                                        minValues = possibleValues;
                                }
                        }
                }
                // Если все клетки заполнены, то завершаем цикл и возвращаем
                   найденное решение
                if (minRow == -1)
                        return true;
                // Иначе если ни одну клетку за проход заполнить не удалось, то
                   завершаем цикл
                else if (1 < minValues.size())
                        break;
        }
        for (auto v : minValues)
        {// Для клетки с минимальным количеством вариантов
                Puzzle solutionCopy = *solution;
                // Пробуем ставить каждое значение по порядку и рекурсивно решать
                solutionCopy[minRow][minColumn] = v;
                if (solveHelper(&solutionCopy, info))
                {// Если решение было найдено, то возвращаем его
                        *solution = solutionCopy;
                        return true;
                }
        }
        // Решение не найдено
        return false;
}

// Получить возможные значения
static Values findPossibleValues(int rowIndex, int columnIndex, const Puzzle&
puzzle, const Puzzle& info)
{
        Values values;
        // Исходно возможны все значения
        for (int i = 1; i < size + 1; i++)
                values.insert(i);
        Values del;
        // Ищем исключения анализируя строки
        del = getRowValues(rowIndex, columnIndex, puzzle, info);
        for (auto d : del)
                values.erase(d);
        // Ищем исключения анализируя столбцы
        del = getColumnValues(rowIndex, columnIndex, puzzle, info);
        for (auto d : del)
                values.erase(d);
        if (values.size() == 1 && *values.begin() == size)
        {// Особый случай - если возможное значение только "X"
                if (rowIndex == 0)
                        if (puzzle[rowIndex + 1][columnIndex] && info[0][columnIndex])
                                if (puzzle[rowIndex + 1][columnIndex] !=
                                info[0][columnIndex])
                                        return Values();
                if (rowIndex == size - 1)
                        if (puzzle[rowIndex - 1][columnIndex] && info[1][columnIndex])
                                if (puzzle[rowIndex - 1][columnIndex] !=
                                info[1][columnIndex])
                                        return Values();
                if (columnIndex == 0)
```

```
                        if (puzzle[rowIndex][columnIndex + 1] && info[2][columnIndex])
                                if (puzzle[rowIndex][columnIndex + 1] !=
                                info[2][columnIndex])
                                        return Values();
                        if (columnIndex == size - 1)
                                if (puzzle[rowIndex][columnIndex - 1] && info[3][columnIndex])
                                if (puzzle[rowIndex][columnIndex - 1] !=
                                info[3][columnIndex])
                                        return Values();
        }
        return values;
}

// Получить исключаемые значения для строк
static Values getRowValues(int rowIndex, int columnIndex, const Puzzle& puzzle,
const Puzzle& info)
{
        Values values;
        if (info[2][rowIndex] && columnIndex < 2)
        {// Если информационный левый столбец имеет значение
                if (columnIndex == 0 || puzzle[rowIndex][0] == size)
                {
                        for (int i = 1; i < size; i++)
                                values.insert(i);
                        values.erase(info[2][rowIndex]);
                }
        }
        if (info[3][rowIndex] && columnIndex > (size - 3))
        {// Если информационный правый столбец имеет значение
                if (columnIndex == (size - 1) || puzzle[rowIndex][size - 1] == size)
                {
                        for (int i = 1; i < size; i++)
                                values.insert(i);
                        values.erase(info[3][rowIndex]);
                }
        }
        // Если в строке уже есть значения
        for (int r = 0; r < size; r++)
                if (puzzle[rowIndex][r] > 0)
                        values.insert(puzzle[rowIndex][r]);
        return values;
}

// Получить исключаемые значения для столбцов
static Values getColumnValues(int rowIndex, int columnIndex, const Puzzle& puzzle,
const Puzzle& info)
{
        Values values;
        if (info[0][columnIndex] && rowIndex < 2)
        {// Если информационная верхняя строка имеет значение
                if (rowIndex == 0 || puzzle[0][columnIndex] == size)
                {
                        for (int i = 1; i < size; i++)
                                values.insert(i);
                        values.erase(info[0][columnIndex]);
                }
        }
        if (info[1][columnIndex] && rowIndex > (size - 3))
        {// Если информационная нижняя строка имеет значение
                if (rowIndex == (size - 1) || puzzle[size - 1][columnIndex] == size)
                {
                        for (int i = 1; i < size; i++)
                                values.insert(i);
                        values.erase(info[1][columnIndex]);
```

```
                        }
                }
                // Если в столбце уже есть значения
                for (int r = 0; r < size; r++)
                        if (puzzle[r][columnIndex] > 0)
                                values.insert(puzzle[r][columnIndex]);
                return values;
        }
};

// Печатает игровое поле
void printPuzzle(const Puzzle& puzzle, const Puzzle& info)
{
        std::cout << "  ";
        for (int i = 0; i < size; i++)
        {
                if (info[0][i])
                        std::cout << (char)(info[0][i] + 64) << " ";
                else
                        std::cout << "  ";
        }
        std::cout << std::endl;
        for (int i = 0; i < size; i++)
        {
                if (info[2][i])
                        std::cout << (char)(info[2][i] + 64) << " ";
                else
                        std::cout << "  ";
                for (int j = 0; j < size; j++)
                {
                        if (puzzle[i][j] == size)
                                std::cout << "X ";
                        else
                                std::cout << (char)(puzzle[i][j] + 64) << " ";
                }
                if (info[3][i])
                        std::cout << (char)(info[3][i] + 64);
                std::cout << std::endl;
        }
        std::cout << "  ";
        for (int i = 0; i < size; i++)
        {
                if (info[1][i])
                        std::cout << (char)(info[1][i] + 64) << " ";
                else
                        std::cout << "  ";
        }
        std::cout << std::endl;
}

int main()
{
        std::ifstream in("input.txt");
        std::ofstream out("output.txt");

        // Чтение из файла
        char c;
        in >> size;
        Puzzle puzzle(size);
        for (int i = 0; i < size; i++)
        {// Чтение доски
                for (int j = 0; j < size; j++)
                {
                        in >> c;
```

```
                puzzle[i].push_back((int)c - 64);
            }
        }
        Puzzle info(4);
        for (int i = 0; i < 4; i++)
        {// Чтение боковой информации
            for (int j = 0; j < size; j++)
            {
                in >> c;
                info[i].push_back((int)c - 64);
            }
        }

        printPuzzle(puzzle, info);
        std::cout << std::endl;

        Puzzle solution = ABCSolver::solve(puzzle, info);
        if (!solution.empty())
        {// Если решение найдено
            printPuzzle(solution, info);
            // Запись в файл
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    if (solution[i][j] == size)
                        out << "X ";
                    else
                        out << (char)(solution[i][j] + 64) << " ";
                }
                out << std::endl;
            }
        }
        else
            std::cout << "There's no solution" << std::endl;

        in.close();
        out.close();
        return 0;
}
```

Листинг 1 – Код программы

5. Реализация программы

Приходящий на вход текстовый файл «input.txt», используемый как пример для проверки кода, выглядит следующим образом.

```
4
@ @ @ @
@ @ @ @
@ @ @ @
@ @ @ @
B @ @ A
@ B @ C
@ @ @ A
@ @ A @
```

Рисунок 1 – Входящий файл 1

Цифра 4 обозначает размер пришедшего поля (квадрат 4×4), символ @ обозначает пустое незаполненное пространство и так как поле имеет размер 4, то и матрица, приходящая в качестве поля, имеет вид матрицы, состоящей из символов @ размером 4×4. Также на вход приходит матрица n×4 (в данном примере 4×4), которая обозначает информацию по границам заполняемой карты.

Текстовый файл, куда заносится результат, после выполнения программы выглядит следующим образом.

```
B C X A
X A C B
C B A X
A X B C
```

Рисунок 2 – Код выходящего файла 1

Для дополнительной проверки также используем входной текстовый файл, содержащий другие параметры поля.

```
6
@ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@ @ E A @ C
@ E D E @ B
E @ B @ C @
C @ D @ B @
```

Рисунок 3 – Входящий файл 2

6 обозначает размер поля 6×6, соответственно входящее начальное поле имеет вид матрицы 6×6, а информация, расположенная по границам заполняемого поля, имеет вид матрицы 6×4.

```
E B X A D C
D C E B X A
B A C X E D
X D B C A E
C E A D B X
A X D E C B
```

Рисунок 4 – Код выходящего файла 2

Консольный вид реализации выглядит следующим образом, и отображает также информацию по краям поля.

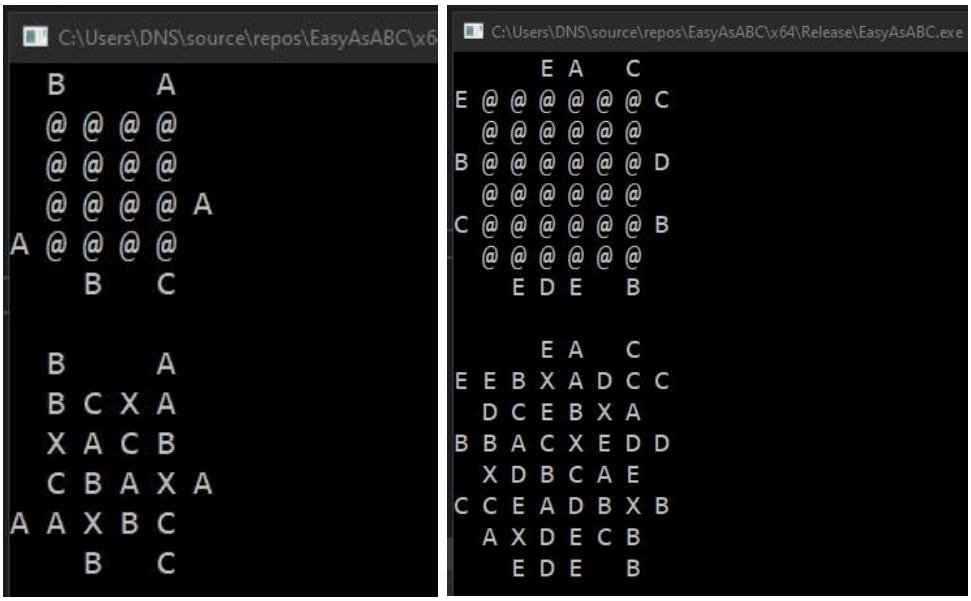

Рисунок 5 – Результат работы программы

Примеры полей брались с сайта головоломок [1], где также проверялась программа на корректность, переводя полученные значения вручную.

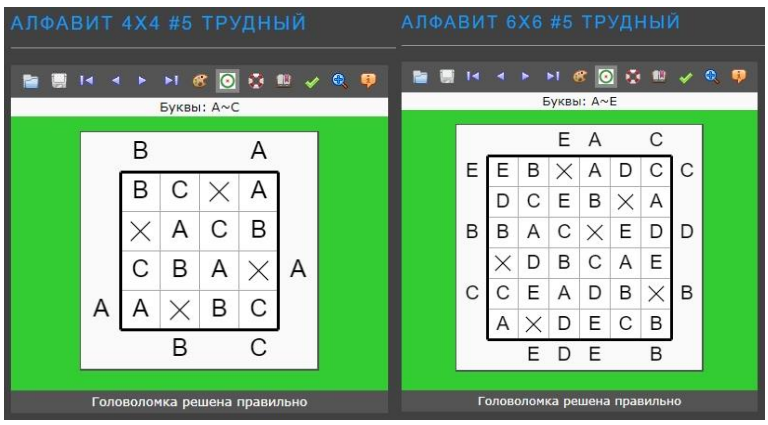

Рисунок 6 – Проверка решения

Заключение

В ходе четвёртой лабораторной работы была создана программа, решающая одну из таких головоломок, как «Первые встречные», которая имеет индивидуальный алгоритм, соответствующий правилам игры. Также программа была протестирована необходимое количество раз для проверки на корректность.

Используемая литература

1. ГОЛОВОЛОМКА «АЛФАВИТ» – http://www.playsudoku.ru/abc.html
2. Головоломка Алфавит (ABC Доку) – https://crossword.nalench.com/abcdoku/
3. Easy as ABC (Mini Puzzles Series #32) – https://www.funwithpuzzles.com/2009/12/abcd-end-view-..
4. Python vs C++: Алгоритм решения судоку – http://itnotesblog.ru/note.php?id=157