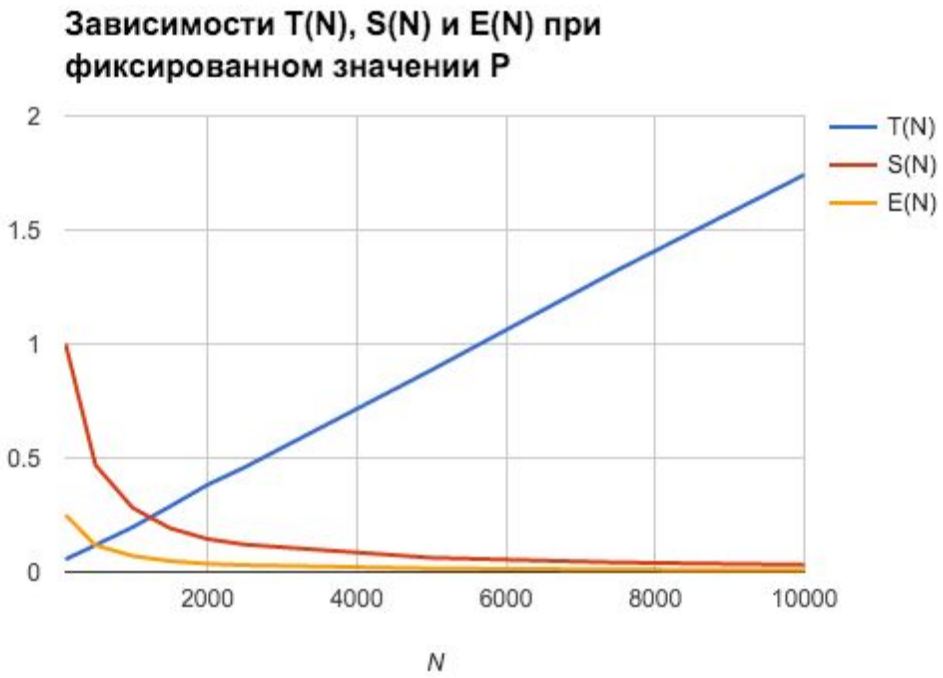
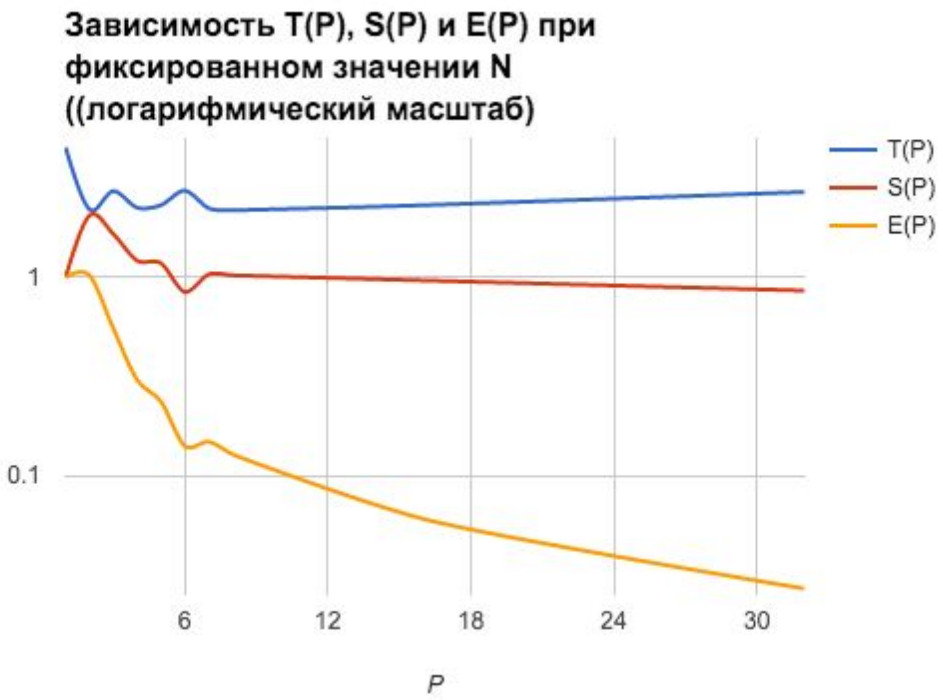
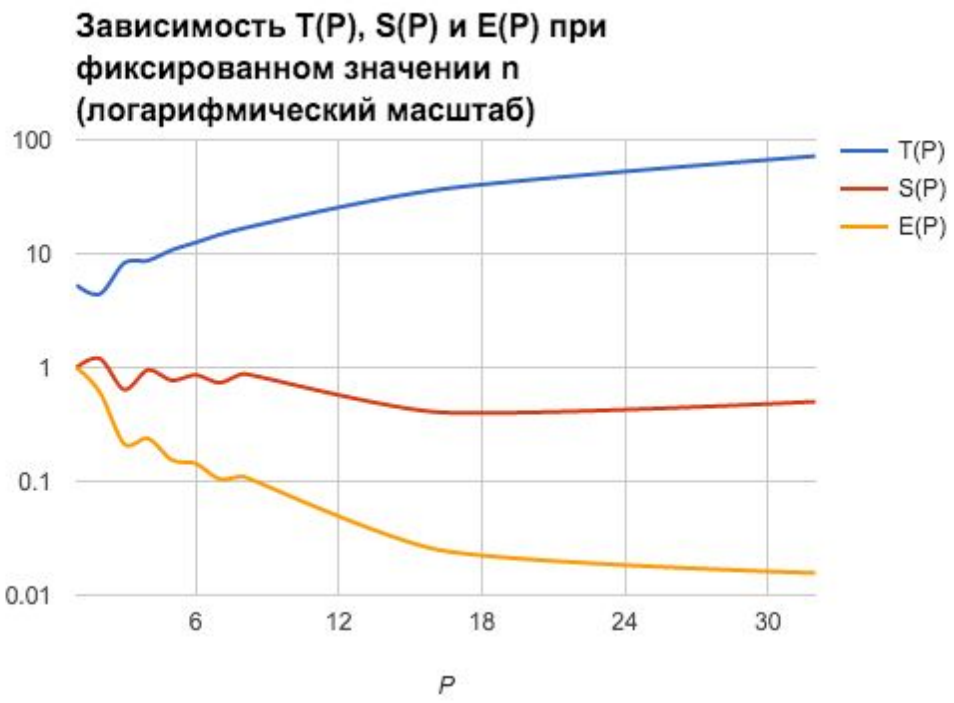
Задание #3. Клеточные автоматы

Горемыкин Александр Олегович, ifresh.wp@gmail.com

ЗАМЕЧАНИЕ: тесты проводились на локальной системе:
Core i5@2.6 (2 cores, hyper-threading up to 4), 8 GB RAM







---

github: https://github.com/sanllier/Practice_Ershov/tree/master/02

```
#include <fstream>
#include <ctime>
#include <vector>

#include "helpers.h"
#include "parparser.h"

//------------------------------------------------------------------------------

typedef vector<char> Rules;

Rules makeRules(int automatNumber) {
    Rules rules(8, 0);

    for (int i = 0; i < 8; ++i) {
        rules[i] = automatNumber % 2;
        automatNumber /= 2;
    }

    return rules;
}

void updateState(vector<char>& state, const Rules& rules) {
    char a = state[0];
    for (int i = 1; i < state.size() - 1; ++i) {
        int b = state[i];
        int c = state[i + 1];
        state[i] = rules[4 * a + 2 * b + c];
```

```
        a = b;
    }
}

void printSate(const vector<char>& state, ostream& oStr) {
    for (int i = 1; i < state.size() - 1; ++i) {
        oStr << int(state[i]) << " ";
    }
}

//---------------------------------------------------------------------------

int main(int argc, char** argv) {

       parparser parser(argc, argv);

    const int automatNumber = parser.get("a").asInt();
    const int localAutomatSize = parser.get("n").asInt();
    const int iterationsThreshold = parser.get("t").asInt();
    const string outFile = parser.get("o").asString();
    const string statFile = parser.get("s").asString();

       //---------------------------------------------------------------------------

       MPICHECK(MPI_Init(&argc, &argv));
       //---------------------------------------------------------------------------

       int commSize = 0;
       int rank = 0;
       MPICHECK(MPI_Comm_size(MPI_COMM_WORLD, &commSize));
       MPICHECK(MPI_Comm_rank(MPI_COMM_WORLD, &rank));

    const int prevProcRank = rank > 0 ? rank - 1 : MPI_PROC_NULL;
    const int nextProcRank = rank < commSize - 1 ? rank + 1 : MPI_PROC_NULL;

    const Rules rules = makeRules(automatNumber);
    std::vector<char> state(localAutomatSize + 2, 0);

    const int localFirstIndex = rank * localAutomatSize;
    const int localLastIndex = localFirstIndex + localAutomatSize - 1;
    const int midIndex = (commSize * localAutomatSize) / 2;

    if (localFirstIndex <= midIndex && midIndex <= localLastIndex) {
        state[midIndex - localFirstIndex + 1] = 1;
    }

    //---------------------------------------------------------------------------

    MPI_Status status;

    const double startTime = MPI_Wtime();

    for (int i = 0; i < iterationsThreshold; ++i) {
        updateState(state, rules);

        MPICHECK(MPI_Sendrecv(&state[1], 1, MPI_CHAR, prevProcRank, 0,
&state[localAutomatSize + 1],
                                        1, MPI_CHAR, nextProcRank, 0, MPI_COMM_WORLD,
&status));

        MPICHECK(MPI_Sendrecv(&state[localAutomatSize], 1, MPI_CHAR, nextProcRank, 0,
&state[0],
                                                        1, MPI_CHAR, prevProcRank, 0,
MPI_COMM_WORLD, &status));
    }
```

```
    const double endTime = MPI_Wtime();

    if (rank == MASTER) {
        auto printHeader = [argc, argv, commSize](ofstream& str) {
            str << "-----------------------TASK 2-----------------------\n";
            for (int i = 0; i < argc; ++i) {
                str << argv[i] << " ";
            }
            str << "on " << to_string(commSize) << " procs.\n\n";
        };

        ofstream statStr(statFile.empty() ? "stat.txt" : statFile, ofstream::out);
        printHeader(statStr);
        statStr << "TOTAL TIME: " << endTime - startTime << "\n";
        statStr << "----------------------------------------------------\n\n";
        statStr.close();

        ofstream oStr(outFile.empty() ? "output.txt" : outFile, ofstream::out);
        printSate(state, oStr);
        for (int i = 1; i < commSize; ++i) {
            MPICHECK(MPI_Recv(&state[0], state.size(), MPI_CHAR, i, 0, MPI_COMM_WORLD,
&status));
            printSate(state, oStr);
        }
        oStr << "\n\n";
        oStr.close();
    } else {
        MPICHECK(MPI_Send(&state[0], state.size(), MPI_CHAR, MASTER, 0,
MPI_COMM_WORLD));
    }

       //--------------------------------------------------------------------------

       MPICHECK(MPI_Finalize());
       return 0;
}
```