**Решение задач для поступления на кафедру Acronis**

**Задача №2**

*Постановка задачи:* Найти число уникальных конечных комбинаций, возникающих в игре крестики-нолики.

*Решение задачи:*

1. Найдем число уникальных комбинаций партий, закончившихся **на 5 ходу**:
2. Всего есть *8* вариантов расположения знаков, при которых партия заканчивается
3. Для каждой такой комбинации три крестика определены однозначно, а оставшиеся *2 нолика* мы можем *расположить в* *любые из оставшихся 6 клеток:*

**В итоге:**

1. **На 6 ходу**:
2. Аналогично пункту **1.a)** возникает множитель *8*
3. Аналогично пункту **2.a)** возникает множитель
4. Однако в этот раз все усложняется тем, что в пункте **2.b)** мы учитываем и расположение 3 крестиков в выигрышной позиции, поэтому из слагаемого полученного из пунктов **2.a)** и **2.b)** нужно вычесть эти комбинации. Найдем это вычитаемое значение:

Т.к. диагональные выигрышные комбинации из крестиков уже не возможны (из-за однозначного наличия комбинации из 3

ноликов), то теперь *коэффициент меняется с 8 на 6*

Для каждой выигрышной комбинации из трех крестиков

остается *2 варианта расположения 1 выигрышной комбинации* из трех ноликов

**В итоге:**

1. **На 7 ходу**:
2. Аналогично пункту **1.a)** возникает множитель 8
3. При конкретном расположении выигрышной комбинации, *3 нолика* можно *расположить в 6 клеток* (без учета того, что они не могут образовывать выигрышную комбинацию)
4. И кроме этого надо *расположить 1 крестик* который может стоять *в одной из 3 клеток*
5. Аналогично пункту **2.c)**, только с ноликами и теперь в каждой из таких комбинаций, оставшийся *1 крестик* можно еще *расположить в одной из 3 клеток.*

**В итоге:**

1. **На 8 ходу**:
2. Аналогично пункту **1.a)** возникает множитель 8
3. При конкретном расположении выигрышной комбинации,

*4 крестика* можно *расположить в 6 клеток* (без учета того, что они не могут образовывать выигрышную комбинацию)

1. И кроме этого надо *расположить 1 нолик* который может стоять *в одной из 2 клеток*
2. Аналогично пункту **2.c)**, только теперь в каждой из таких комбинаций, оставшийся *1 нолик* можно еще *расположить в одной из 3 клеток и* оставшийся *1 крестик в одной из 2 клеток*

**В итоге:**

1. На 9 ходувсе усложняется. После нужно будет посчитать еще ничейные комбинации, а также основные вычисления так просто уже не проходят. **Выигрышных уникальных конечных комбинаций на 9 ходу**:
2. В этом случае, когда мы рассматриваем диагональные выигрышные комбинации и после их рассмотрения ставим 2 крестика так, что на поле получаются 2 выигрышные комбинации, а потом так же делаем с вертикальными или горизонтальными выигрышные комбинациями, то учитываем одну и ту же комбинацию 2 раза. Всего *диагональных выигрышных комбинаций 2*, а *горизонтальных и вертикальных 6.*
3. В связи с пунктом **5.a)** при подсчете, количество конечных комбинаций, которые на доске будут давать 2 выигрышных комбинации, будем делить пополам (т.к. учитываем их 2 раза, в подсчете диагональных вариантов и вертикальных с горизонтальными).
4. Так же стоит отметить то, что в данном случае нам стоит расположить только крестики, т.к. нолики потом будут расположены однозначно, и то, что в данном пункте мы будем считать сразу те комбинации, которые не приведут к наличию на поле 2 выигрышных комбинаций (одной из крестиков и другой из ноликов)
5. При определенном расположении выигрышной комбинации нам нужно *расположить 2 крестика на 6 полей*, это число равно:
6. Для каждой из диагональных комбинаций существует *7* вариантов расположения двух крестиков, при которых конечные комбинации будут учтены дважды, тогда это количество будет делится на *2*.
7. Аналогично для вертикальных выигрышных комбинаций: только там *количество таких вариантов будет равно 5*. Но также надо учесть, что оставшиеся 2 крестика нужно расположить так, чтобы не получилось комбинации из 3 ноликов, т.е. теперь всего вариантов:

**В итоге:**

1. **Теперь рассмотрим ситуацию с ничьими:**
2. Первым делом докажем, что в ничейной комбинации обязательно имеются два нолика стоящие рядом в одной вертикали или горизонтали. Докажем для горизонтали, т.к. тогда для вертикали все решается простым поворотом или симметричным отображением.

*Доказательство*: Пусть это не так, тогда неизбежна такая:

, либо такая: , либо такая: .

Все остальные получаются симметричным отображением и поворотом.

**В первом** варианте, нужно закрыть одну выигрышную комбинацию и с помощью одного нолика можно это сделать лишь одним способом, при котором как раз получается ничейная комбинация и два нолика стоят рядом в одной горизонтали.

**Во втором** варианте нет способов одним ноликом закрыть две диагонали и не получить при этом выигрышной комбинации из трех ноликов.

**В третьем** варианте есть три способа расположить нолик, все они находятся на 2 вертикали, и могут располагаться на первой, второй и третьей горизонталях и в любом из этих вариантов получается ничейная комбинация, в которой два нолика стоят рядом в одной горизонтали.

1. Теперь рассмотрим два паттерна, при которых может получится ничейная комбинация (все остальные получаются симметрией и поворотом):

, .

1. Рассмотрим *первый паттерн* и пронумеруем позиции от 1 до 9 таким образом:



Тогда будем рассматривать расположение последнего нуля только в верхнюю часть (нижнюю рассматривать смысла нет из-за симметрии)

Расположение нолика в позиции 1 заставляет нас ставить

следующий ноль в позицию 9, что дает выигрышную диагональную комбинацию из 3 нулей, что для нас невозможно. ***×***

А вот постановка первого нолика в позицию 2 заставляет нас

ставить второй ноль так же в 9 позицию и в конечном итоге дает ничейную комбинацию. ***✓***

Постановка нолика в позицию 3 приводит к единственному варианту постановки последнего нуля в позицию 9 и образует ничейную комбинацию ***✓***

1. Теперь рассмотрим второй паттерн и пронумеруем поля аналогично.

Расположение нолика в месте, где нет пересечения двух

красных линий невозможно, т.к. тогда придется ставить

оставшийся нолик на пересечение трех линий, а такового в этом случае нет. ***×***

Постановка одного из нулей в позицию 3 заставляет ставить второй нолик в позицию 5, и аналогично наоборот, остается только один вариант, это расположение одного из нулей в позицию 1, а второго в позицию 6. Таким образом получили еще 2 комбинации с учетом, того, что найденную можно отобразить симметрично. ***✓✓***

**В итоге:**

1. На самом деле количество ничьих плюс количество выигрышей на 9 ходу можно найти намного проще. **Выигрышных и ничейных уникальных конечных комбинаций на 9 ходу**:
2. Всего вариантов расстановок, при которых на поле находятся 4 нолика и 5 крестиков
3. Аналогично вычитаем все варианты, при которых три нолика в конечной расстановке составляют выигрышную комбинацию. Таковых:

**В итоге:**

***В конечном итоге всего уникальных комбинаций:***

**Для проверки полученных результатов написан (на скорую руку) класс, приведенный ниже:**

**TicTacToeCounter.h**

#pragma once

#include <map>

#include <set>

const int X = 1;

const int O = 2;

const int onlyWins = 1;

const int exactMoves = 2;

using namespace std;

class TicTacToeCounter

{

private:

int \_amountOfComb;

int \_amountOfDraws;

set<map<int, int>> \_comb;

int \_amountOfUniqDraws;

int \_currFlag;

int \_lastStep;

bool SortedByExactMoves(map<int, int> currCollocation, int currDepth);

void SetSignByNumberOfMove(int numberOfMove, int& where);

bool SortedByWinsOnly(int currDepth, map<int, int> currCollocation);

void PrintCombination(map<int, int> tmp);

void RecursiveCount(int currDepth, map<int, int> currCollocation, int flag);

public:

void PrintAllInformation();

void PrintCombinations();

set<map<int, int>> getUniqComb() { return \_comb; };

int getAmountOfUniqDraws() { return \_amountOfUniqDraws; };

int getAmountOfDraws() { return \_amountOfDraws; };

int getAmountOfUniqComb() { return (int)\_comb.size(); };

void CountCombinations(int lastStep, int flag);

void Reset();

void CountUniqDraws();

bool HorizontalIsWinning(map<int, int>& t, int i);

bool VerticalIsWinning(map<int, int>& t, int i);

bool DiagonalIsWinning(map<int, int>& t);

bool ThisCollocIsWinning(map<int, int>& t);

TicTacToeCounter();

~TicTacToeCounter();

};

**TicTacToeCounter.cpp**

#include "TicTacToeCounter.h"

#include <iostream>

using namespace std;

void TicTacToeCounter::CountCombinations(int lastStep, int flag){

Reset();

\_currFlag = flag;

\_lastStep = lastStep;

map<int, int> currCollocation;

for (int i = 1; i <= 9; i++)

currCollocation.insert(make\_pair(i, 0));

RecursiveCount(0, currCollocation, flag);

CountUniqDraws();

}

void TicTacToeCounter::Reset() {

\_amountOfComb = 0;

\_amountOfDraws = 0;

\_comb.clear();

\_amountOfUniqDraws = 0;

\_currFlag = 0;

}

void TicTacToeCounter::CountUniqDraws() {

for (auto item : \_comb) {

if (!ThisCollocIsWinning(item)) {

\_amountOfUniqDraws++;

}

}

}

bool TicTacToeCounter::HorizontalIsWinning(map<int, int>& t, int i) {

return (t[1 + 3 \* i] == t[2 + 3 \* i] && t[2 + 3 \* i] == t[3 + 3 \* i] && t[3 + 3 \* i] != 0);

}

bool TicTacToeCounter::VerticalIsWinning(map<int, int>& t, int i) {

return (t[1 + i] == t[4 + i] && t[4 + i] == t[7 + i] && t[7 + i] != 0);

}

bool TicTacToeCounter::DiagonalIsWinning(map<int, int>& t) {

return ((t[1] == t[5] && t[5] == t[9] && t[9] != 0) || (t[3] == t[5] && t[5] == t[7] && t[7] != 0));

}

bool TicTacToeCounter::ThisCollocIsWinning(map<int, int>& t) {

for (int i = 0; i < 3; i++) {

if (HorizontalIsWinning(t, i) || VerticalIsWinning(t, i)) {

return true;

}

}

if (DiagonalIsWinning(t)) {

return true;

}

return false;

}

bool TicTacToeCounter::SortedByExactMoves(map<int, int> currCollocation, int currDepth) {

if (ThisCollocIsWinning(currCollocation)) {

\_comb.insert(currCollocation);

\_amountOfComb++;

return true;

}

else if (currDepth == (\_lastStep - 1)) {

for (int i = 1; i <= 9; i++) {

if (currCollocation[i] == 0) {

SetSignByNumberOfMove(\_lastStep, currCollocation[i]);

if (!ThisCollocIsWinning(currCollocation)) {

\_amountOfDraws++;

}

\_comb.insert(currCollocation);

currCollocation[i] = 0;

}

}

\_amountOfComb++;

return true;

}

return false;

}

inline void TicTacToeCounter::SetSignByNumberOfMove(int numberOfMove, int& where) {

if (numberOfMove % 2 == 1)

where = X;

else

where = O;

return;

}

inline bool TicTacToeCounter::SortedByWinsOnly(int currDepth, map<int, int> currCollocation) {

if (ThisCollocIsWinning(currCollocation) && currDepth != \_lastStep)

return false;

if (ThisCollocIsWinning(currCollocation) && currDepth == \_lastStep) {

\_comb.insert(currCollocation);

\_amountOfComb++;

return true;

}

return true;

}

void TicTacToeCounter::PrintCombination(map<int, int> tmp) {

for (int i = 1; i <= 9; i++) {

if (tmp[i] == X) {

cout << "x ";

}

if (tmp[i] == O) {

cout << "o ";

}

if (tmp[i] == 0) {

cout << "\_ ";

}

if (i % 3 == 0) {

cout << endl;

}

}

cout << endl;

}

void TicTacToeCounter::PrintAllInformation(){

cout << "Step " << \_lastStep << ": " << endl;

cout << "combinations = " << this->\_amountOfComb << endl;

cout << "uniq comb = " << this->getAmountOfUniqComb() << endl;

if (\_currFlag == exactMoves) {

cout << "draws = " << this->\_amountOfDraws << endl;

cout << "uniq draws = " << this->\_amountOfUniqDraws << endl;

}

return;

}

void TicTacToeCounter::PrintCombinations() {

for (auto item: \_comb) {

PrintCombination(item);

}

}

void TicTacToeCounter::RecursiveCount(int currDepth, map<int, int> currCollocation, int flag) {

if (flag == exactMoves) {

if (SortedByExactMoves(currCollocation, currDepth))

return;

}

else {

if(SortedByWinsOnly(currDepth, currCollocation))

return;

}

for (int i = 1; i <= 9; i++) {

if (currCollocation[i] == 0) {

SetSignByNumberOfMove(currDepth + 1, currCollocation[i]);

RecursiveCount(currDepth + 1, currCollocation, flag);

currCollocation[i] = 0;

}

}

return;

}

TicTacToeCounter::TicTacToeCounter(){

}

TicTacToeCounter::~TicTacToeCounter(){

}