МІНІСТРЕСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

Кафедра теорії оптимальних процесів

Магістерська робота

на тему:

«Функція Шпрага-Гранді у комбінаторній теорії ігор»

Затверджено на

Засіданні кафедри теорії оптимальних процесів

Протокол №_____ від _____

Зав. кафедри

_____ проф. Бартіш М.Я.

Виконав:

студент групи ПМа-51м

Мацех Маркіян Олегович

Науковий керівник

доц. Дудзяний І.М.

Львів 2012

Завдання на дипломну роботу

Завдання на дипломну роботу полягає у детальному огляді теорії Шпрага-Гранді, її застосуванні до рівноправних ігор; наведенні основних теоретичних висновків у цій галузі; демонстровації алгоритмів розв’язку рівноправних ігор на прикладах; і як кінцевий результат - розробці програмного пакету як підтвердження викладеної теорії.

Реферат

У цій роботі розглядається теорія рівноправних ігор і шляхи вирішення задач цієї теорії. Починається робота зі вступу до теорії ігор, пізніше увага надається комбінаторній теорії ігор, а потім, за допомогою орієнтованих графів, визначаються самі рівноправні ігри. З самого початку наведено декілька прикладів для пояснення суті таких ігор, не вдаючися до способів їх розв’язку.

Після цього, надається увага проблемам в такій теорії і способу їх вирішення. Починається це з гри Нім, з якої, завдяки декільком гіпотезам, і виводиться алгоритм розв’язку рівноправних ігор. Як буде показано в самій роботі, суть теорії Шпрага-Гранді полягає у зведенні усіх рівноправних ігор до гри Нім і рекурсивному обчисленні значень Шпрага-Гранді для всіх можливих ходів у грі.

Після наведення теоретичної частини, буде продемонстровано програмний пакет для розв’язування деякого класу ігор, серед яких «Шахи Доусона», «Королева», «Хрестики-Хрестики» та інші.

Зміст

1. Теоретичні відомості

1.1 Вступ

Все наше життя складається з неперервного прийняття рішень – коли ми обираємо обід в ресторані, коли ми обираємо книжку для прочитання, коли шахіст вирішує який хід зробити, коли футболіст вирішує на кого передати пас, коли бізнесмен вирішує чи погоджуватися на пропозицію, зрештою, коли ми обираємо чому приділити свій час.

Не кожен з нас задумується, над тим, що кожне рішення викреслює всі інші варіанти розвитку подій. Ніхто не знає, який з обраних варіантів насправді привів би до кращого результату, тому що фактично можна побачити результат тільки одного вибору. Але в багатьох областях діяльності прийняття рішень піддається аналізу, і результати аналізу допомагають відповідальній людині зробити вибір, який принесе їй найбільше користі. Інколи результат являє собою великі суми грошей, інколи корисність проведеного часу, інколи навіть людські життя.

Часто так буває, що для того щоб досягти мети, потрібно зробити декілька кроків(прийняти декілька правильних рішень). В такому випадку вибирати стає складніше, бо варіантів розвитку подій набагато більше, і вони ще й переплітаються між собою. Зробивши перший вибір не найкращим, умови для другого будуть ще складніші, і кінцевий результат можливо й не вдасться досягти. Але водночас, якщо не дивитися на завдання в повному обсязі, а тільки на найближчий результат, то попри те, що кожен наступний вибір буде здаватися оптимальним в даному контексті, він буде віддаляти нас від поставленого завдання. Тому появилося таке поняття як оптимальна стратегія, яка являє собою обдуману послідовність прийняття рішень. До оптимальних стратегій змушені вдаватися бізнесмени, політики, менеджери і будь-хто, чия професія заставляє дивитися на декілька кроків вперід.

Особливим випадком прийняття рішень є *умови конфлікту,* тобто такі умови, в яких наші рішення, а точніше їх наслідки, переплітаються з рішеннями інших людей, які в свою чергу будуть впливати на наші наступні кроки. Це дуже поширена ситуація, і насправді, якщо задуматися – будь-які наші дії впливають на інших людей, деякі менше, деякі більше.

Якщо сумістити багатокрокрокові прийняття рішень і умови конфлікту, то виходить дуже цікавий обєкт для аналізу, який в науці назвали Теорією Ігор. Цікавим моментом є те, що назва ця не надумана, і крім того має багатосторонній зміст. Справді, більшість відомих ігор підпадають під це визначення, але сюди також і входять дуже складні життєві ситуації, як політичні баталії, біологічні процеси, економічні закони та ін. Іронія в тому, що, якою б серйозною не було ситуація, для математики вона всеодно залишиться лише грою.

1.2 Короткі історичні відомості

Теорія ігор бере свій початок у 1713 році, у листі Джеймса Вальдегрейва. З того часу вона досить пасивно розвивалася, поки в XX столітті Джон Фон Нейман опублікував свою роботу у 1928. В другій половині XX століття теорія ігор почала дуже активно розвиватися завдяки таким особистостям як Джон Неш, Рейнхард Зелтен, Джон Харсаньї, Джон Мейнард Сміт та ін. Забігаючи наперід, відмітимо той факт, що Теорія Шпрага-Гранді, яка розглядається в цій, роботі була розроблена у 1935-1939 роках, перед такими відомими результатами, як Ситуація Рівноваги Неша(Nash Equilibrium), Баєсівськими іграми та ін.

У 2001 році Теорія Ігор істотно популяризувлися завдяки фільму Рона Говарда “Ігри Розуму” (“Beautiful Mind”), в якому описується життя математика Джона Неша.

На сьогоднішній день теорія ігор активно розвивається. У березні 2012 року Стенфордський університет організував масові безплатні онлайн-курси навчання,

в яких однією з найважливіших дисциплін була Теорія Ігор. Це не може не свідчити про широке використання цієї теорії.

Цікавим фактом щодо Теорії Ігор є те, що вона, на відміну від більшості математичних наук, націлена не на вирішення задач фізики, а в основному на задач економіки. Хоча, з часом коло питань, які підлягали аналізу в цій теорії розширилося до військової справи, медицини, соціального прогнозування, питання моралі, масової поведінки індивідів тощо.

Саме тому ця робота зконцентрована на дослідження теорії ігор, а конкретніше рівноправних ігор.

1.3 Комбінаторна теорія ігор

Комбінаторна теорія ігор – це розділ теорії ігор, в якому розглядаються *послідовні* ігри *двох гравців* з *повною інформацією*. Почнемо з означення комбінаторної гри і гри в загальному:

Грою називають систему прийняття рішень в конфліктній ситуації.

Означення 1.1 *Кобмінаторною грою* називають гру, яка задовільняє наступним умовам:

1. Є два гравці
2. Є множина, зазвичай скінченна, ситуацій гри
3. Для кожного гравця є множина можливих ходів
4. Гравці ходять по черзі
5. Гра закінчується, коли гравець, чия черга ходити, не може зробити хід
6. Гра закінчується за скінченне число ходів
7. Інформація про ходи гравців і правила гри відома обом гравцям.

Є декілька класифікацій комбінаторних ігор:

1. Кобмінаторні ігри можна грати *нормальним* способом, або в *піддавки*. Нормальний спосіб - коли гравець, який повинен робити хід, не може цього зробити – програє. У Піддавках він виграє. Піддавки важче піддаються аналізу і є менш поширеними.
2. Комбінаторні ігри бувають скінченні і нескінченні. Скінченна гра закінчується за скінченну кількість кроків, як би гравці в неї не грали. Нескінченна гра може мати набір ходів, які циклічно переходять один в одного, і тому, при бажанні гравців, може продовжуватися вічно. Зазвичай на практиці обмежують кількість однакових повторюваних ходів і домовляються про нічию, для обмеженості часу гри.
3. Рівноправні і партизанські. В рівноправних іграх множина ходів ніяк не залежить від гравця, а в партизанських – залежить(наприклад в шахах – білі

/чорні фігури). Фактично в рівноправних іграх вся різниця між гравцем 1 і гравцем 2 полягає в тому, що гравець 1 ходить перший.

4. Ігри залежні/незалежні від інших умов. Бувають ігри, в яких певна множина ходів дозволена тільки після 6ої вечора, або тільки після спеціального ходу противника, або ще від інших зовнішніх умов. Такі ігри є набагато складнішими і важче піддаються аналізу. В даній роботі такі ігри не розглядаються.
5. Ігри з нічиєю/без нічиї. Перші – це ті, в яких є тільки два варіанти закінчення гри – виграє перший гравець або виграє другий гравець. В іграх з нічиєю допускається третій варіант – не виграє жоден з гравців.

В цій роботі ми вивчаємо теорію Шпрага-Гранді, яка застосовна до рівноправних скінченних ігор, в яких виключається нічия. Тобто при будь-якому розвитку подій, якийсь з гравців програє за скінченну кількість кроків, а саме – коли в нього не буде ходів. Крім того, нас цікавить в основному нормальний хід гри(а не піддавки), хоч багато розглянутого матеріалу застосовно і до піддавок. Тому далі при наведенні матеріалу буде допускатися, що мова йде саме про такі ігри, якщо не вказано інакше, і звертатися до них будемо як до рівноправних.

1.4 Рівноправні ігри

Насправді, визначити рівноправні ігри можна багатьма способами, наприклад за допомогою графів, ігор з перевертанням монет та багатьма іншими. Зараз ми наведемо теорію рівноправних ігор, визначену за допомогою графів, а також проілюструємо декілька ігор.

Для початку дамо визначення напрямленого графа в констексті рівноправних ігор

Означення 1.2 В контексті комбінаторних ігор *орієнтованим графом* G називають пару *(X,F)* , де *X*- непорожня множина вершин, а F – функція, яка для кожного $x \in X$ повертає підмножину $X, F(x) \subset X$. Кожна вершина відображає *позицію(стан)* в грі, а функція *F(x)* для заданого *x* повертає множину позицій, в які

можна перейти з *x* (їх називають *послідовниками x*). Якщо *F(x)* повертає порожню множину, то *x* називають кінцевою(термінальною) позицією.

Означення 1.3 Комбінаторною грою на орієнтованому графі називають гру, яку грають за такими правилами:

1. Гравець №1 ходить першим, з позиції x_0
2. Гравці ходять по черзі
3. В позиції *x*, гравець, чия черга ходити, вибирає хід у позицію з множини *F(x)*
4. Гравець, який опинився у кінцевій позиції на свій хід, і отже не може походити – програє

З такого означення слідує, що гра може продовжуватися нескінченно, а оскільки нас цікавлять скінченні ігри, то потрібно обмежити поле гри скінченною кількістю ходів. Для цього введемо наступні поняття.

Означення 1.4 Шляхом в орієнтованому графі називають послідовність $x_0, x_1, \dots, x_m$, де $x_i \in F(x_{i-1})$ для всіх $i = 1, \dots, m$

Довжиною шляху називають кількість вершин, через які він проходить = *m+1*

Означення 1.5 Циклом в орієнтованому графі називається шлях $x_0, x_1, \dots, x_m$, де $x_0 = x_m$ і $x_0, x_1, \dots, x_{m-1}$ – різні вершини, $m \geq 1$

Означення 1.6 Орієнтований граф називається *обмеженим*, якщо для будь-якої точки $x_0 \in X$ існує таке число *n* (можливо залежне від x_0), що будь-який шлях з x_0 має довжину меншу або рівну *n*

Якщо множина *X* скінченна, то обмежений граф називають ациклічним.

Тепер означимо основний обєкт дослідження цієї роботи

Означення 1.7 *Рівноправною* грою називають комбінаторну гру на орієнтованому графі, коли граф обмежений

Повертаючися до класичного означення орієнтованого графу, можемо встановити такі відповідності: вершинами в ньому є стани гри, а ребрами – переходи з одного стану в інший. Ребро з вершини x_0 до вершини x_1 існує, якщо $x_1 \in F(x_0)$

Тепер можна перейти до класифікації станів. Оскільки нічиї немає, то всі стани поділяють на два типи.

Означення 1.8 Стани рівноправної гри визначаються як:

1. N –стан (next player) – стан, виграшний для гравця, який зараз буде ходити
2. P – стан (previous player) – стан виграшний для гравця, який щойно походив

Будемо казати, що вершина графу є N,P-станом(позицією) гри, якщо поточний стан(позиція) припадає на цю вершину в графі.

Нескладно побачити, що NP-стани можна визначити рекурсивно з термінальних станів.

1. Позначимо всі термінальні вершини як P-стани.
2. Знайдемо всі вершини, які мають послідовників, які є P-станами, і позначимо їх як N-стани.
3. Позначимо вершини,чиї послідовники є тільки N-станами, як P-стани
4. Якшо в кроці (3) P-станів не знайдено – зупинитися, інакше – до кроку 2.

Отже, маємо ще один спосіб (рекурсивний) для визначення чи вершина є N-станом, чи P-станом.

Означення 1.8' Стани рівноправної гри можна визначити рекурсивно:

1. Всі термінальні вершини є P-станами
2. З кожного N-стану є *хоча б один* хід у P-стан
3. З кожного P-стану всі ходи ведуть тільки в N-стан.

Для наочності наведемо приклад простої гри на графі, і визначимо її NP-позиції.

Приклад 1 Маємо наступну гру на графі, з пронумерованими вершинами (рисунок 1). Визначити її *NP*-позиції

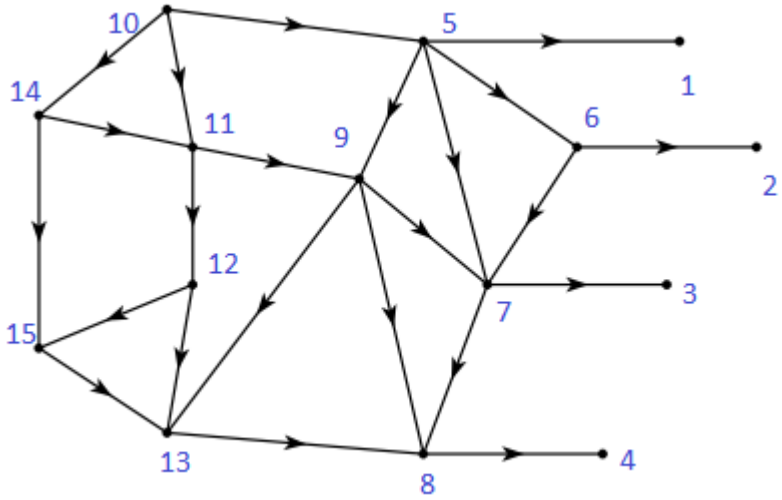


Рисунок 1. Звичайно гра на графі

Почнемо з термінальних позицій. Очевидно, що їх 4: у вершинах 1,2,3,4. Позначаємо їх як *P*-позиції. Тепер розглянемо всі вершини, що мають серед послідовників термінальні вершини. Це вершини 5,6,7,8. Позначаємо їх як *N*-позиції. Тепер шукаємо всі вершини, всі послідовники яких є відомими *N*-позиціями. Наразі можна відзначити тільки одну таку вершину – 13, позначаємо її *P*-станом. Вершини 9,10 нам не підходять, бо серед їх послідовників є вершини (13,10), стан яких невідомий на момент вибору. Тепер продовжуємо алгоритм, повертаючись до кроку 2, і знову шукаємо *N*-стани. Отримуємо вершину 9. Продовжимо алгоритм вибору ітеративно і отримаємо такі результати

1. *P*: 1,2,3,4
2. *N*: 5,6,7,8
3. *P*: 13
4. *N*: 9,15,12
5. *P*: 11
6. *N*: 10,14

В результаті отримаємо такі списки

P: 1, 2, 3, 4, 11, 13

N: 5, 6, 7, 8, 9, 10, 12, 14, 15.■

Завдання аналізу рівноправних ігор – визначити до якого класу належить поточний стан, і крім того *виявити закономірності* залежності *NP*-стану від ситуації в грі. Тобто можливість передбачити переможця в будь-якій ситуації.

Визначення термінальних точок можна перефразувати на зрозумілішу мову

Якщо гравець в *N*-стані, тобто в виграшному становищі, то найдеться принаймні один перехід в програшний стан, в якому опиниться його противник. Якщо ж гравець в *P*-стані, тобто програшному становищі, то всі його ходи будуть вести в *N*-стан,в якому ходити буде його противник.

Фактично, можемо дійти висновку, що якщо гравець в *N*-стані – він, при бажанні, може в ньому залишатисядо кінця гри, і зрештою виграти. Якщо гравець в *P*-стані, то при мудрій грі противника він ніяк не вийде з цього стану і зрешто програє. Єдина його надія на виграш – це те, що противник не знає оптимальної стратегії, і помилково з *N*-стану перейде знову у *N*-стан. Тоді гравці поміняються ролями, і той хто спочатку був в *P*-позиції тепер буде в *N*-позиції і зможе виграти.

Ось ми і вивели оптимальну стратегію для гравця – завжди залишатися в *N*-стані. Але як це зробити? Як сказати чи дана ситуація є *N*-станом чи *P*-станом? І який з ходів приведе гравця в *P*-стан? Відповіді на ці питання і дає теорія Шпрага-Гранді. Але перед тим як до неї перейти, потрібно розглянути ще декілька понять.

1.5 Найпростіша рівноправна гра

Для кращого розуміння рівноправних ігор розглянемо один з найпростіших видів цих ігор, відомий під назвами палички, або віднімашки(Substraction games). Ця гра стала відомою завдяки французькому телевізійному шоу Ford Boyard. Суть гри проста: на столі є *n* паличок, кожен з гравців по черзі забирає 1,2 або 3 палич-

ки зі столу. Гра закінчується, коли гравець не може зробити хід, тобто коли на столі не залишилося паличок.

Проведемо простий аналіз такої гри. Позначимо як S_n – ситуацію в грі, де залишилося n паличок. Термінальною позицією є позиція, в якій лишилося 0 паличок на столі, тобто S_0. Це є P-позиція. В неї ми можемо попасти з позицій S_1, S_2, S_3, які є в свою чергу N-позиціями. Звернемо увагу на те, що з S_2 і S_3 можна перейти і не в виграшну позицію, а наприклад в S_1, але нас цікавлять оптимальні стратегії, і згідно означення – позиція вважається N-позицією якщо існує хоча б один перехід в P-позицію, а такий перехід є – в стан S_0. Підемо далі: стан S_4 – це P-стан, оскільки будь-який хід з нього приведе в N-стан (S_1, S_2, S_3). Провівши далі подібний аналіз, можна побачити закономірність: для позиції S_0, S_4, S_8, S_{12}... – P-позиції, решта – N-позиції, і ми хочемо залишатися в них , шоб виграти гру.

Цю гру можна зобразити і на графі:

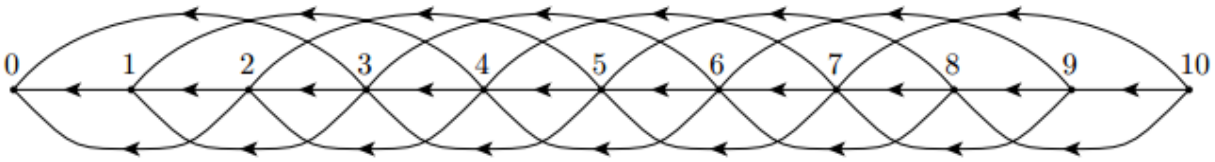


Рисунок 2. Гра в палички для n=10

Цікавим нюансом в цій грі, є те, що новачок зазвичай думає «я почну грати, а в ході гри зрозумію яка виграшна стратегія». А насправді весь хід гри залежить від першого ходу. Якщо гравець попав хоч раз в P-позицію, то, він вже з неї не вийде.

1.6 Гра Нім

Гра Нім є найвідомішою, і ключовою грою в рівноправних іграх (це ми невдовзі доведемо). Дамо її визначення

Означення 1.9 Грою Нім називають рівноправну гру з наступними правилами:

1. Є N купок з $x_1, x_2, \ldots x_N$ одиницями деякого предмету (далі будемо вважати що це монетки)

2. За один хід гравець може взяти будь-яку позитивну кількість монеток з однієї купки
3. Виграє той, хто забрав останню монетку з останньої купки

Проаналізуємо цю гру. Очевидно- термінальною позицією є позиція (0,…,0). Гра з одною купкою – тривіальна. Достатньо просто забрати всі монетки і виграш забезпечений.

Для гри з двома купками *P*-позиціями будуть ті, в яких розміри купок еквівалентні, тобто (1,1), (2,2) і т.д. Справді, якшо черга суперника ходити – він змінить цю рівновагу, а перший гравець знвоу її відновить. Так він за скінченну кількість кроків прийде до ситуації (0,0), тобто до виграшу.

Якщо купок 3, ситуація стає складнішою. Очевидно, (1, 1, 1), (1, 1, 2), (1, 1, 3) і (1, 2, 2) є всі *N*-позиціями тому що з них можна перейти в (1, 1, 0) або (0, 2, 2). Наступна найпростіша позиція - $(1, 2, 3)$, тому що з неї можна перейти тільки в стані перераховані раніше, отже це – *P*-позиція. Далі продовжувати аналіз в такому дусі стане складніше, бо глибина рекурсії буде збільшуватися великими темпами.

Для подальшого аналізу гри Нім і їй подібних, введемо поняття Нім-Суми.

Означення 1.10 Нім-сумою двох додатніх натуральних чисел називають побітову xor-суму цих чисел.

Це означення потребує додатково пояснення. Кожне натуральне число має представлення у двійковому вигляді, тобто

$$x = x_m 2^m + x_{m-1} 2^{m-1} + \cdots + x_1 2 + x_0, \forall x \in N \qquad (1)$$

Де кожне x_i – 1 або 0,а m залежить від самого числа. Наприклад $22 = 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = (10110)_2$. Нім-сума рахується шляхом розкладання чисел в такий вигляд і операцією побітного додавання а модулем 2(xor-сума), а потім перетворенням результату з 2ового в 10вий.

З таким уточненням можна ввести допоміжне означення Нім-суми

Означення 1.10' Нім-сумою чисел $(x_m \cdots x_0)_2$ і $(y_m \cdots y_0)_2$ є $(z_m \cdots z_0)_2$, і ми кажемо, що

$(x_m \cdots x_0)_2 \oplus (y_m \cdots y_0)_2 = (z_m \cdots z_0)_2$, де для всіх $k, z_k = x_k \oplus y_k$, тобто $z_k = 1$, якщо $x_k + y_k = 1$, і $z_k = 0$ в інших випадках.

наприклад

$$\begin{gathered} 22 = 10110_2 \\ 51 = 110011_2 \\ \hline \text{нім} - \text{сума} = 100101_2 = 37 \end{gathered}$$

Нім-сума володіє більшістю властивостей, що й звичайне додавання – асоціативність, комутативність, має нейтральний елемент (0) і обернений елемент – саме це число. Тепер прийшов час застосувати знання про нім-суми до важливих теоретичних висновків.

Теорема 1(Бутона) У Німі позиція $(x_1, x_2, \ldots, x_N)$ є *P*–позицією *тоді і тільки тоді*, коли нім-сума усіх розмірів купок рівна нулю.

Доведення. Позначимо через *P* множину Нім позицій з нім-сумою рівною нулю, і через *N* – решту ситуацій, тобто такі ситуації, нім-сума яких більша нуля. Перевіримо 3 умови з означення **(1.8`)**

1. *Всі кінцеві позиції є в P.*Єдина термінальна позиція є $0 \oplus 0 \oplus 0 \ldots \oplus 0 = 0$, і ця позиція входить в *P*.
2. *Для кожної позиції в N є перехід в позицію в P*. Збудуємо такий перехід наступним чином. Подивимося на нім-суму як на додавання колонок. Вибираємо найлівішу(найзначущішу) колонку з непарною кількістю 1(одиничок). Тепер вибираєм будь-який рядок, який має одиничку в цій колонці і міняємо всі його розряди так, щоб кількість одиничок в кожній колонці була парна. Після цього число монеток в вибраній купці зменшиться (тому що 1 в найбільш значущому полі міняється на нуль), і крім цього загальна нім-сума буде рівною нулю. І отже – такий перехід цілком легальний.

3. *Кожен перехід з P веде в N.* Якщо ми в P, і $x_1 \oplus x_2 \oplus ... \oplus x_N = 0$,і ми поміняємо x_1 на $x_1' < x_1$, то попередня сума $x_1' \oplus x_2 \oplus ... \oplus x_N$ вже не може бути рівною 0, що й доводить, що новоутворена позиція є N-станом.

З цих трьох властивостей виплаває, що P є множиною P-станів, а N – множиною N-станів.■

Приклад 1 Розглянемо наступну гру Нім $(x_1, x_2, x_3) = (13,12,8)$. Порахуємо її нім-суму

$$\begin{array}{r} 13 = 1101 \\ 12 = 1100 \\ 8 = \ \ 1000 \\ \hline \text{німсума} = 1001 = 9 \end{array}$$

Оскільки $9 \neq 0$, то це N-позиція. Якщо трошки придивитися, то можна навіть знайти виграшний перехід. Потрібно замінити одне з чисел так, щоб нім-сума була рівною нулю. Одною з таких замін є заміна 13 на 9, тобто відбирання 4 монет з першої купки.

$$\begin{array}{r} 9 = \ \ 100 \\ 12 = 1100 \\ 8 = \ \ 1000 \\ \hline \text{німсума} = 0000 = 0 \end{array}$$

Також можна відняти 7 монет з купки з 12а монетами, залишивши там 5 монет. Насправді, доведення теореми дає значні підказки для вибору наступного числа, і можна слідувати простому правилу – шукати заміну вибраному число як німсуму решти купок монет.

1.7 Теорія Шпрага-Гранді (теорема про еквівалентність кожної гри Німу)

Тепер ми переходимо до найважливішої частини дипломної робота – теорія Шпрага-Гранді. Перед основною теоремою роботи, потрібно розглянути допоміжну лему, відому під назвою «Лема про нім зі збільшеннями». Для цього дамо наступне означення

Означення 1.11 Нім зі збільшеннями - це гра, яка у всьому подібна до звичайного німу, але з одним вийнятком – додається ще один хід. Замість віднімання деякої кількості монеток, їх можна додати.

Лема 1(про збільшення) Нім зі збільшенням повністю еквівалентний звичайному німу.

Доведення здійснюється дуже просто. Для цього нам всього лиш потрібно продемонструвати такий наочний приклад – якщо гравець замість віднімання монеток, додає нові – то його противнику достатньо зробити симетричний хід – відняти таку ж суму з тої купки. Тоді ситуація повністю повертається до такої яка була 2 ходи назад.■

Суть цієї теореми в тому, що додавання монеток не допоможе гравцю при оптимальній грі противника, і тому вся гра «Нім зі збільшеннями» не має ніякого сенсу – вона еквівалентна Німу.

Тепер час перейти до найважливішої теореми в цій роботі.

Теорема 2(Шпрага-Гранді) Розглянемо деякий стан S рівноправної гри. З нього є переходи в деякі стани $S_i, i = 1 \dots k, k \geq 0$. Тоді, стану S цієї гри можна поставити у відповідність купку Німа деякого розміру x, яка буде повністю описувати стан нашої гри. Це число x називають значенням Шпрага-Гранді стану S.

Крім того, x можна знайти наступним рекурсивним способом: підрахуємо x_i для кожного переходу S_i, і тоді виконується

$X = mex\ (x_1, x_2, \dots, x_k)$, де mex – функція, яка повертає найменше ціле невід'ємне значення,яке не належить її множині-аргументу.

Таким чином, почавши з термінальних вершин можна порахувати значення Шпрага-Гранді для всіх станів нашої гри. Якщо значення функції рівне нулю – це P-стан, інакше – N-стан

Доведення теореми здійснюється по індукції.

Для термінальних вершин, згідно теореми $x = mex(\emptyset) = 0$, що цілком правильно, термінальні позиції – *P*-позиції.

Тепер розглянемо стан *S*, з якого є переходи. Припущення індукції – що всі S_i, в які ми можемо перейти, вже пораховані.

Порахуємо величину $p = mex(x_1, x_2, \dots, x_k)$. Тоді, згідно визначення функції *mex*, ми можемо зробити висновок, що для кожного числа i в проміжку $[o, p)$ знайдеться хоча б один відповідний перехід в якийсь зі станів S_i. Крім того, можуть існувати і додаткові переходи в стани, в яких значення Гранді більше ніж p.

Це означає, що поточний стан еквівалентний стану Німа з збільшеннями з купкою розміру *P*: насправді, в нас є переходи із поточного стану в стани з купками всіх менших розмірів, а так само можуть бути і переходи в стани з купками більшими розмірів.

Отже, величина $mex(x_1, x_2, \dots, x_k)$ насправді є шуканим значенням Шпрага-Гранді для поточного стану , що й треба було довести. ■

Що нам дає ця теорема? Фактично ми отримали алгоритм розв'язку будь-якої рівноправної гри! Звісно, є ще декілька нюансів, але всі розв'язки зводяться до цього підходу – рекурсивного відшукання значень Шпрага-Гранді. Також з цієї теореми можна виділити означення, з яким ми будемо часто зустрічатися в цій роботі.

Означення 1.12 Функцію, яка кожному стану гри *S* ставить у відповідність число Шпрага-Гранді *x* називають функцією Шпрага-Гранді

$$F(S) = x = mex(x_0, x_1, \dots, x_N) = mex(F(S_0), F(S_1), \dots, F(S_N))$$

де $S_i, i = 0, \dots, N -$ всі можливі переходи в грі зі стану S

На даному етапі можна описати попередній алгоритм розв'язку рівноправної гри, який застосовний для вузького кола задач:

На вхід маємо поточний стан гри і її правила. Щоб сказати чи це *N*-позиція чи *P*-позиція потрібно:

1. Виписати всі можливі переходи в інші стани
2. Порахувати значення Шпрага-Гранді для цих переходів(застосувати пункти 1-2-3 для переходів)
3. Порахувати значення *mex* для цих значень
4. Якщо отримане значення рівне нулю, то це *P*-позиція, інакше *N*-позиція

Проілюструємо роботу алгоритму на раніше розглянутому прикладі з грою на графі(приклад 1)

Приклад 2 Для гри з прикладу 1 визначити чи вершина номер 9 є *N* чи *P* позицією використовуючи функцію Шпрага-Гранді.

1. Всі переходи в нас вже виписані, оскільки це гра на графі. З вершини №9 можна перейти у вершини 7, 8 і 13.
2. Рахуємо значення Шпрага-Гранді для вершини №7
    a. З цієї вершини є переходи в вершини 3 і 8
    b. Рахуємо значення Шпрага-Гранді для вершини №3
        i. Переходів немає – це термінальна вершина. Повертаємо 0
    c. Рахуємо значення Шпрага-Гранді для вершини №8
        i. З цієї вершини є перехід в вершину 4
        ii. Рахуємо значення Шпрага-Гранді для вершини №4
            1. Переходів немає – це термінальна вершина. Повертаємо 0
        iii. Більше переходів немає – повертаємо *mex*(0) = 1
    d. Більше переходів немає – повертаємо *mex*(0, 1) = 2
3. Рахуємо значення Шпрага-Гранді для вершини №8
    a. Це значення ми вже порахували. Повертаємо 1
4. Рахуємо значення Шпрага-Гранді для вершини №13
    a. З цієї вершини є перехід у вершину 8
    b. Рахуємо значення Шпрага-Гранді для вершини №8

i. Це значення ми вже порахували. Повертаємо 1

c. Більше переходів немає - повертаємо *mex*(1) = 0

5. Більше переходів немає – повертаємо *mex*(2, 1, 0) = 3

6. Отримане значення = 3 > 0 , робимо висновок що це *N*-позиція.

Для повноти прикладу проілюструємо граф зі всіма порахованими значеннями Шпрага-Гранді

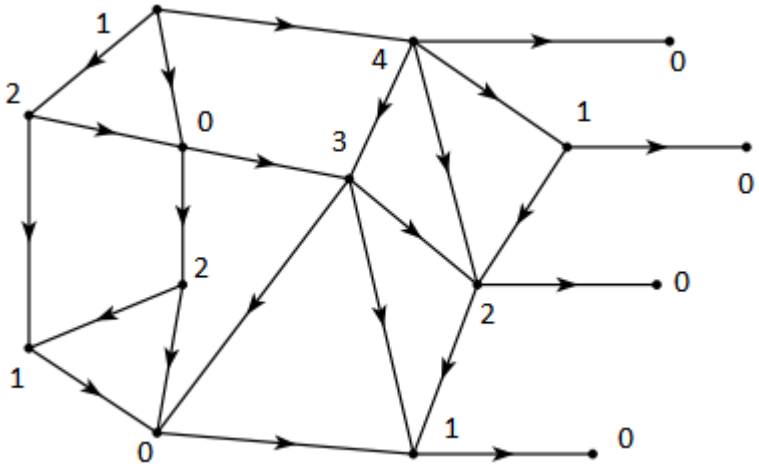


Рисунок 3. Пораховані значення Шпрага-Гранді для гри з прикладу 1

Як бачимо – глибина рекурсії навтьі для такої простої гри досягає 4 вкладень. При складніших іграх і більших значеннях *n* глибина рекурсії стає неприпустимою для обчислень

Цей алгоритм не застосовний до всіх рівноправних ігор, тому що в деяких іграх перехід з деякого стану може вести до *суми ігор*, а не просто до однієї гри. Поняття суми ми розглянемо в наступному розділі магістерської роботи, після чого запишемо повноцінний алгоритм.

1.8 Суми ігор

Поняття гри можна узагальнити до суми ігор наступним чином. Припустім ми маємо декілька ігор, в кожній свій початковий стан. Хід гравця заключається в тому, щоб вибрати будь-яку з ігор, зробити в ній хід(за правилами цієї гри) і лишити решту ігор в такому ж стані. Гра закінчується коли в кожній грі досягнули

кінцевої позиції, і отже немає більше ходів. Гравець, який зробив останній хід виграє. Формалізуємо це визначення

Означення 1.13 Нехай в нас є n обмежених графів $G_1 = (X_1, F_1), G_2 = (X_2, F_2), \dots, G_n = (X_n, F_n)$. Сумою $G_1 + G_2 + \cdots + G_n$ n графів називають граф $G = (F, X)$, який визначається наступни чином: множина вершин X є декартовим добутком множин вершин кожної гри, $X = X_1 \times X_2 \dots \times X_n$, а функція F для вершини $x = (x_1, x_2, \dots, x_n)$ визначається так:

$$F(x) = F(x_1, x_2, \dots, x_n) = F_1(x_1) \times \{x_2\} \times \dots \times \{x_n\}$$
$$\cup \{x_1\} \times F_2(x_2) \times \dots \times \{x_n\}$$
$$\cup \dots$$
$$\cup \{x_1\} \times \{x_2\} \times \dots \times F_n(x_n)$$

Означення 1.14 Сумою n ігор на графах називають гру на графі, який є сумою графів цих ігор

Тепер розглянемо теорему, яка дозволяє застосовувати попередній підхід Шпрага-Гранді до сум ігор.

Теорема 3 Якщоg_i – це функція Шпрага-Гранді гри G_i, то $G = G_1 + G_2 + \cdots + G_n$ є таку функцію Шпрага-Гранді: $g(x_1, \dots, x_n) = g_1(x_1) \oplus g_2(x_2) \oplus \dots \oplus g_n(x_n)$.

Доведення аналогічне доведенню теореми 2.

Практичний висновок теореми – значення Шпрага-Гранді суми ігор шукається через нім-суму значень Шпрага-Гранді цих ігор. Нескладно побачити, що більшість ігор можна виразити як суму подібних, але менших ігор. Цей підхід дає нам багато можливостей для оптимізацій.

Тепер, після визначення суми ігор, можна виписати повноцінний алгоритм визначення стану гри, а точніше відшукання значення Шпрага-Гранді:

1.9 Алгоритм розв’язку рівноправної гри

1. Виписати всі можливі переходи в інші стани

a. Якщо перехід веде в одну гру – порахувати значення Шпрага-Гранді звичайний способом(виконати кроки 1-2 цього алгоритму)
b. Якщо перехід веде в в суму ігор – порахувати значення Шпрага-Гранді як нім-суму значень Шпрага-Гранді цих ігор

2. Порахувати значення *mex* для цих значень
3. Якщо отримане значення рівне нулю, то це *P*-позиція, інакше *N*-позиція

Продемонструємо роботу алгоритму на наступній задачі.

Приклад 3 (Гра хрестики-хрестики)

Розглянемо клітчасту полоску розміром $1 \times n$ клітинок. За один хід гравцю потрібно поставити один хрестик, але заборонено ставити хрестики в сусідні поля. Програє той, хто не може зробити хід. Сказати хто виграє при оптимальній грі.

Розв'язок. Є 2 варіанти як гравець може поставити хрестик – посередині поля або скраю. Якщо хрестик поставлено скраю (в клітинці 1 або n) то гра зводиться до цієї ж гри, але з кількістю клітинок рівною n – 2. Якщо ж хрестик пставлено посередині, в клітинку i, де $1 < i < n$, то гра розпадається на суму двох ігор: з кількістю клітинок $i-2$, і кількістю клітинок $n-i-1$. Тобто множина переходів в стани складається з всеможливих комбінацій хрестиків на середині поля, і випадку на краю поля. Таким чином отримуємо наступну формулу для розрахунку значення Шпрага-Гранді:

$$g(n) = mex\left\{g(n-2), \bigcup_{i=2}^{n-1} g(i-2) \oplus g(n-i-1)\right\} \blacksquare$$

Після самодостатньої теоретичної частини, ми можемо перейти до практичної : розв'язування задач і їх програмування.

2. Програмна Реалізація

2.1 Архітектура проекту

Програма була написана мовою C#, як консольна аплікація. На вхід вона приймає вибір однієї з ігор, потім її параметри(для кожної гри вони різні), і поточний стан, з чого робить висновок чи це є *N*, чи *P* стан. Також для деяких ігор програма знаходить закономірність розв'язків, а також підказує оптимальний хід. Результати виводяться на екран консолі в одному з двох виглядів:

1. У вигляді таблиці залежності результату вді стану гри
2. Читабельною англійською мовою (українську мову в консолі виводити значно важче)

Як вказано на малюнку нижче програма складається з 4 основних компонент

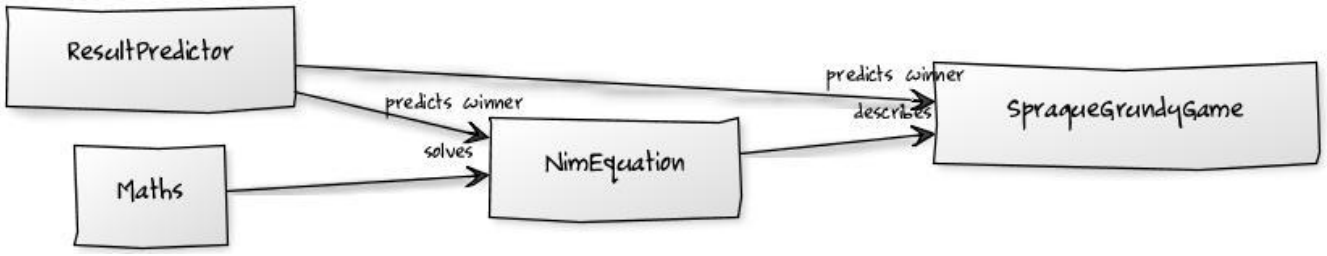


Рисунок 4. Архітектура програми

Перша – основна частина – сам обєкт гри(SpragueGrundyGame). Він задає правила гри, логіку переходів станів, і власне дає можливості решті компонентів робити їхню роботу.

Наступний – NimEquation – описує гру Шпрага-Гранді за допомогою Нім-рівняння. Цей компонент використовується не у всіх іграх, а тільки в тих, де потрібно знайти оптимальний хід.

Для розв'язку Нім-рівнянь використовується модуль Maths, в якому прописані нескладні математичні алгоритми.

І остання логічна частина програми – ResultPredictor. Його роль – з результатів обчислень передбачити який з гравців виграє у даній грі, а також показати це в одному з вищезгаданих варіантів.

Нижче наведено структуру проекту даної програми

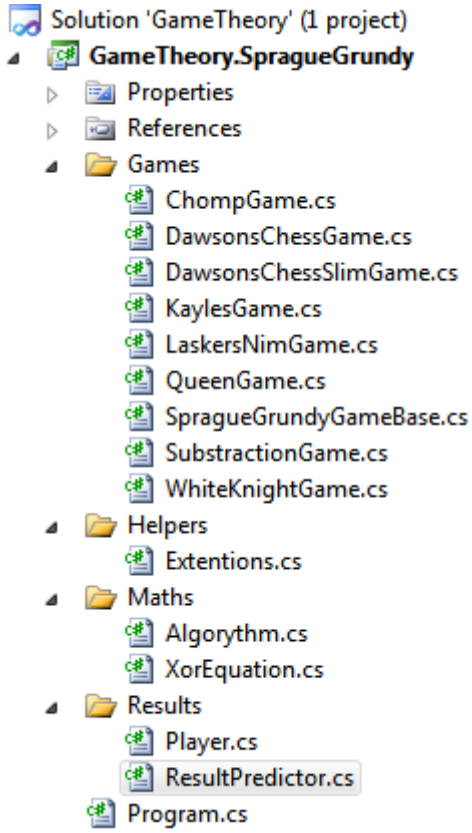


Рисунок 5. Структура проекту

Тепер перейдемо до самих ігор і їх програмного розв'язку. Демонстрація програми буде наводитися в наступному порядку: спочатку буде розглянуто структуру коду, а потім кожна гра по черзі буде розглядатсия за шаблоном «гра, математична модель, код». Інколи гра буде зілюстрована графом, коли це матиме сенс.

2.2 Технічний Опис

Оскільки програмна реалізація і алгоритм не є складними, і всі ігри відрізняються лише переходами, то вибрано модель обєктно орієнтованого дизайну, в якій буде визначено базовий абстрактний клас гри зі всім кодом, який перевикористовується в кожній грі поокремо, і на успадковані класи покладається відповідальність задати логіку переходів по станах, а також умова зупники рекурсії, тобто фактично визначення термінальних вершин.

Нижче продемонстрований код для базовго класу гри Шпрага-Гранді

```
public abstract class Sprague Grundy GameBase<TKey>
{
   protected Sprague Grundy GameBase()
   {
      RecursionCount = 0;
      CachedRecCount = 0;
   }

   public int RecursionCount { get; private set; }
   public int CachedRecCount { get; private set; }

   public void ResetCounters()
   {
      RecursionCount = 0;
      CachedRecCount = 0;
   }

   public int CachedObjects { get { return _cache.Count; } }

   private readonly Dictionary<TKey, uint> _cache = new Dictionary<TKey, uint>();

   public uint SGValue(TKey key)
   {
      RecursionCount++;

      uint grundyValue;

      if (TryGetCachedValue(key, out grundyValue))
         return grundyValue;

      if (TryStopRecursion(key, out grundyValue))
         return grundyValue;

      CachedRecCount++;
      grundyValue = Algorythm.Mex(GetSGValuesForTransitions(key));

      CacheValue(key, grundyValue);

      return grundyValue;
   }

   protected abstract bool TryStopRecursion(TKey key, out uint value);

   protected abstract HashSet<uint> GetSGValuesForTransitions(TKey key);
```

```
        public virtual HashSet<TKey> GetStateTransitions(TKey key)
        {
          throw new IOException("The given game doesn't provide state transitions view, only sprague-grundy values");
        }

        private void CacheValue(TKey key, uint grundyValue)
        {
          _cache[key] = grundyValue;
        }

        private bool TryGetCachedValue(TKey key, out uint value)
        {
          return _cache.TryGetValue(key, out value);
        }
      }
```

Оглянемо коротко методи та поля цього класу.

Перший оголошений член класу – захищений конструктор, який будуть неявно викликати нащадки. Він ініціалізує поля *RecursionCount* і *CachedRecCount* в нуль. Ці поля потрібні для замірювань швидкості, глибини рекурсії і ефективності коду. Як видно далі – вони доступні ззвоні для читання, але закриті для модифікації. Є єдиний метод, яким можна змінити ці значення ззовні - *ResetCounters*(), і він встановлює їх в 0. Це надає додатковий рівень безпеки – компілятор не дозволить ніякого несанкціонваного доступу до цих полів, а отже заміри будуть точними.

Далі йдуть засоби кешування даних. Пізніше ми покажемо наскільки суттєвим є кешування для швидкодії, у скільки разів воно зменшує глибину рекрусії і чому воно є необхідним. Також розглянемо структуру даних використану для цього. Наразі достатньо сказати, що це приватна змінна, яка доступна ззовні тільки для перегляду, моніторингу ситуації. До цієї змінної мають доступ тільки два методи: CacheValue і TryGetCachedValue відповідно для запису і читання з кешу. Останній повертає значення true або false в залежності від того чи є в кеші

даний ключ. Якщо є, то в змінну value передається адреса на знайдене значення, і з місця виклику функції можна отримати цю адресу.

Тепер перейдемо до головного моменту в класі – метод SGValue, який приймає на вхід ключ – стан в грі. Зауважимо, що весь клас є узагальненим (generic), власне через те, що стан в грі може задаватися різними наборами значень. Інколи достатньо звичайного числа, інколи потрібно декілька чисел, а інколи й складніші структури даних. Тому значення Шпрага-Гранді буде шукатися по різному в залежності від цього типу даних. Щодо самої логіки відшукання значення: перш за все ми пробуємо взяти значення з кешу. Це спірне питання, першо дією могла б бути спроба зупинити рекурсію. Але після численних спроб і реалізації різних ігор було вирішено, що спроба зупинити рекурсію може займати значно довше ніж спробува витягнути значення з кешу. Крім того рекурсію припиняти доводиться рідше ніж вичитувати з кешу, і тим більше – більшість випадків закінчення рекурсії закешовуються при перших декількох звертаннях. Якщо не вдалося отримати закешоване значення і ми не дійшли до термінальних вершин, тобто умова зупинки рекурсії не спрацювала, то ми насправді обчислюємо шукане значення, тобто шукаємо *mex* від значень Шпрага-Гранді всіх можливих переходів. Тут же враховується, що сума ігор в результаті також поверне число, що полегшило реалізацію функції *mex*, і дозволило місцями оптимізувати час виконання. Власне цей рядок коду і запускає рекурсивний процес обчислення. Множина значень Шпрага-Гранді для всіх переходів є особливою для кожної гри, тому метод GetSGValuesForTransitions зроблений абстрактним, щоб нащадки реалізували цю частину логіки.

Після обчислення значення ми закешовуємо його, з причин наведених вище.

Лишився лише один нерозглянутий момент цього класу – метод GetStateTransitions. Деякі нащадки реалізують переходи в стани оптимально, і таким чином не дають можливості переглянути переходи в стани – вони повертають тількьи значення Шпрага Гранді. Інколи буває потрібно і переглянути множину можливих пере-

ходів, для відладки наприклад. Для цього і використовується цей метод. Він робиться абстрактним, тому що це не завжди потрібно. Але тоді користувач бібліотеки має бути повідомлений, що цей метод не зреалізваний в поточній грі. Це робиться шляхом піднімання помилки з відповідним повідомленням. За звичних умов метод GetSGValuesForTransitions мав би для кожного стану з методу GetStateTransitions ставити в вівдповідність його значення Шпрага-Гранді. Але інколи оптимізації це забороняють.

Крім того, в коді визначено декілька допомагаючих класів, які роблять код читабельнішим.

Тепер можемо перейти до конкретних ігор.

2.2.1 Гра «Палички» (substraction game)

Ми вже описували правила цієї гри в розділі «найпростіша рівноправна гра», і навіть ілюстрували її на графі. Тепер продемонструємо математичну модель і код реалізації функції GetSGValuesForTransitions. Оскільки гра тривіальна, то й реалізація проста. Щоправда ми реалізуємо загальніший випадок гри – коли можна брати від 1 до m паличок зі стола.

1. Математична модель

$$\forall i, i \geq 0, \text{якщо } i \; mod \; m = 0, \text{то це } P - \text{позиція}, \text{інакше} - N - \text{позиція}$$

2. Код

```
var set = new HashSet<uint>();
foreach (var substraction in _substractionSet)
{
   if (n < substraction)
      continue;
   set.Add(SGValue(n - substraction));
}
return set;
```

2.2.2 Гра Хрестики-Хрестики

Правила гри а також математичну модель цієї гри ми описали в прикладі 3 з розділу «Алгоритм розв'язку рівноправної гри». Тепер тільки продемонструємо програмний код методу GetSGValuesForTransitions:

```
var set = new HashSet<uint>();

  set.Add(SGValue(key - 2));
  for (uint i = 2; i <= key - 1; i++)
    set.Add(SGValue(i - 2) ^ SGValue(key - i - 1));

  return set;
```

2.2.3 Гра Нім Ласкера (Lasker's Nim)

Правила гри наступні: маємо n купок монеток заданих розмірів. За один хід гравець може взяти будь-яку ненульову кількість монет з будь-якої купки, або ж розділити купку на дві непусті. Програє той хто не може зробити хід.

Повторюючи аналогічні до попереднього прикладу дії можемо вивести таку математичну модель переходу станів

$$g(i) = mex\left\{\bigcup_{i=0}^{n-1} g(i), \bigcup_{i=1}^{n-1}(g(i) \oplus g(n-i))\right\}$$

і реалізацію в коді

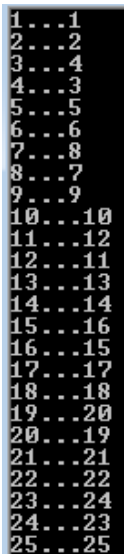
```
protected override HashSet<uint> GetSGValuesForTransitions(uint n)
{
  var set = new HashSet<uint>();

  for (uint i = 0; i <= n - 1; i++)
    set.Add(SGValue(i));

  for (uint i = 1; i <= n - 1; i++)
    set.Add(SGValue(i) ^ SGValue(n - i));

  return set;
}
```

Прогнавши код для n=25 отримуємо такі результати, з яких очевидною стає закономірність



$$g(n) = \begin{cases} n, n\ mod 4 = \{1,2\} \\ n \pm 1, n\ mod\ 4 = \{3,0\} \end{cases}$$

2.2.4 Гра Кеглі (The Game of Kayles)

Правила гри наступні: є n кегель, виставлених в ряд(подібно як в боулінгу, але ряд один). За один удар гравець може вибити або одну кеглю (припускається що гравець настільки вправний, що діаметр кулі йому не завада), або дві кеглі, котрі стоять поруч. Виграє той, хто збив останню кеглю.

Розв'язок, знову ж таки, доволі подібний до попередніх задач. Скільки б кегель він не збив (1 чи 2) – гра розпадається на суму двох незалежних ігор :

$$g(n) = mex\left\{\bigcup_{i=0}^{n-1}(g(i) \oplus g(n-1-i)), \bigcup_{i=0}^{n-2}(g(i) \oplus g(n-2-i))\right\}$$

Нижче маємо відповідний код:

```
protected override HashSet<uint> GetSGValuesForTransitions(uint key)
{
   var set = new HashSet<uint>();

   for (uint i = 0; i <= key - 1; i++)
```

```
            set.Add(SGValue(i) ^ SGValue(key - 1 - i));

        for (uint i = 0; i <= key - 2; i++)
            set.Add(SGValue(i) ^ SGValue(key - 2 - i));

        return set;
    }
```

2.2.5 Гра Ферзь (Королева, Corner The Queen)

Правила гри наступні: маємо шахову дошку, на ній єдина фігура -ферзь. Хід гравця – посунути ферзя куди-небудь за правилами шахів, але тільки на північ, захід, або північний захід. Виграє той гравець який перший став в кут дошки.

Оптимальна стратегія – тримати противника в такій позиції, щоб він не міг ступити в край дошки, або загнати нас в таку позицію. Стан гри тут задається двома параметрами – *n*,*m*. Врахувавши правила руху ферзя можемо вивести таку формулу для обрахунку значень Шпрага-Гранді:

$$g(n,m) = mex\{\bigcup_{i=1}^{n-1} g(n-i,m) \cup \bigcup_{i=1}^{\min(n,m)-1} g(n-i,m-i) \cup \bigcup_{i=1}^{m-1} g(n,m-i)\}$$

Тобто значення Шпрага-Гранді рахується як *mex* всіх можливих ходів ферзя. Програмно це відображається так:

```
protected override HashSet<uint> GetSGValuesForTransitions(Coordinate key)
    {
        var set = new HashSet<uint>();

        int x = key.X, y = key.Y;

        for (int i = 1; i <= x-1; i++)
            set.Add(SGValue(new Coordinate(x - i, y)));

        var northWestBound = Math.Min(x, y);
        for (int i = 1; i < northWestBound; i++)
            set.Add(SGValue(new Coordinate(x - i, y - i)));

        for (int i = 1; i <= y-1; i++)
            set.Add(SGValue(new Coordinate(x, y - i)));
```

```
        return set;
    }
```

Після запуску такої програми і форматування результату в бінарні дані отримаємо такі висновки(для n,m=25)

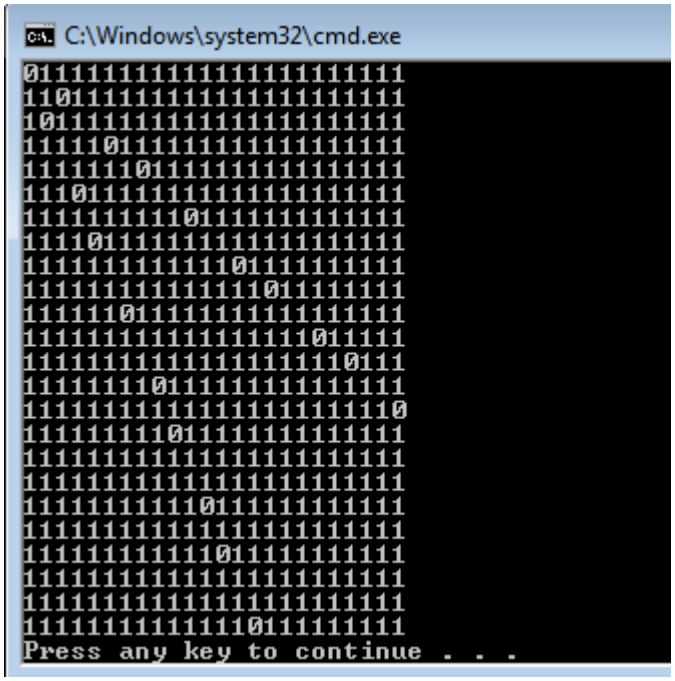


Рисунок 6. Результати розв'язку гри «Королева»

На рисунку 6 зображено матрицю одиничок і нулів у відповідності до полів на шаховій дошці. Починаючи гру з клітинки випадкової n,m ми зможемо або обрати виграшний хід, або ж опинитися в програшному стані. Оптимальною стратегією в грі є тримати противника завжди в вершинах зі значенням 0.

2.2.6 Гра Білий Кінь (White Knight)

Правила гри наступні: маємо шахову дошку розміром $m \times n$. На полі єдина фігура – кінь. Гравці ходять ним по черзі за правилами шах(2 клітинки по одній осі, 1 клітинка по іншій, і так у всеможливих комбінаціях напрямків), але тільки у північному, західному, або північно-західному напрямку з допустими відхиленням в 1 клітинку від заданого напряму. Програє той, хто не може зробити хід. Тобто програшні – крайні верхні 4 клітинки поля.

Для заданої гри є 4 можливих ходи в загальному випадку і 1, 2 або 3 ходи в випадках наближеності до країв. Запишемо загальну формулу:

$$g(n,m) = mex\{g(n-1,m-2), g(n-2,m-1), g(n-2,m+1), g(n+1,m-2)\}$$

Де $n - i > 0, m - i > 0, i = 1,2$

Тепер продемонструємо робочий код і ілюстрацію обчислень:

```
protected override HashSet<uint> GetSGValuesForTransitions(Coordinate key)
 {
    var set = new HashSet<uint>();
    var possibleMoves = new[]
                    {
                       new Coordinate(key.X - 1, key.Y - 2),
                       new Coordinate(key.X - 2, key.Y - 1),
                       new Coordinate(key.X - 2, key.Y + 1),
                       new Coordinate(key.X + 1, key.Y - 2)
                    };
    foreach (var move in possibleMoves.Where(move => move.X > 0 && move.Y > 0))
       set.Add(SGValue(move));
    return set;
 }
```

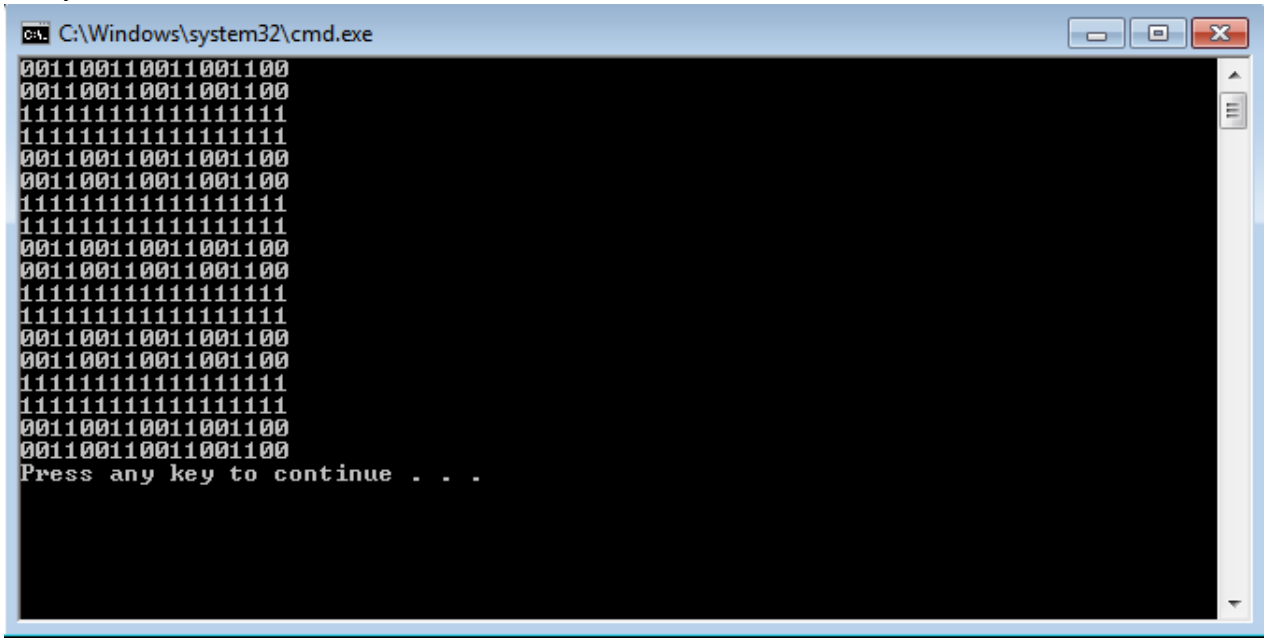


Рисунок 7. Результати обчислення гри Білий Лицар

2.2.7 Гра Шахи Доусона (Dawson's Chess)

Правила гри наступні: маємо шахову дошку $1 \times n$, на ній в першому ряді виставлені білі пішаки, в третьому – чорні. Хід гравця полягає в виборі кольору і русі пішака вперід по одній з клітинок. Коли гравець посунув пішака вперід, то два поля біля цього пішака стають не доступними для ходу, оскільки пішака, який туди стане, одразу поб'ють фігурою іншого кольору. Програє той, хто не може походити пішаком так, щоб його не збили.

Розв'язок цієї задачі значно цікавіший ніж попередніх, адже на ньому ми проілюструємо важливість теореми про суму ігор. Ця гра еквівалентна грі з купками, де можна забирати 3 монетки з купки, або ділити купку на дві частини. Тоді правила гри можна перефразувати наступним чином 1)можна забрати 1 монетку з купки, якщо це вся купка, 2) можна забрати 2 монетки з будь-якої купки, 3) можна взяти 3 монетки з будь-якої купки і атким чином розділити її на дві частини(кожна з яких може виявитися порожньою)

Продемонструємо зреалізований алгоритм в коді:

```
public override HashSet<PileList> GetStateTransitions(PileList key)
{
    var set = new HashSet<PileList>();

    // remove 1 chip if it's a whole pile
    for (int i = 0; i < key.Count; i++)
    {
        if (key[i] == 1)
        {
            var list = new PileList(key);
            list.RemoveAt(i);
            set.Add(list);
        }
    }

    // remove 2 chips from any pile
    for (int i = 0; i < key.Count; i++)
    {
        var list = new PileList(key);
```

```
            list[i] -= 2;
            if (list[i] <= 0)
                list.RemoveAt(i);
            set.Add(list);

        }

        // remove 3 chips and split this pile on 2, 1 or 0 (depending on situation)
        for (int i = 0; i < key.Count; i++)
        {
            var list = new PileList(key);
            list[i] -= 3;

            int leftChipsCount = list[i];
            int middleOfChips = leftChipsCount / 2 + 1;

            list.RemoveAt(i);

            if (leftChipsCount == 0)
            {
                if (!list.IsEmpty())
                    set.Add(new PileList(list));
                else
                    set.Add(new PileList(list) { leftChipsCount });
            }
            else if (leftChipsCount > 0)
                set.Add(new PileList(list) { leftChipsCount });

            for (int j = 1; j < middleOfChips; j++)
                set.Add(new PileList(list) { j, leftChipsCount - j });
        }

        return set;
    }
```

Одразу на цій грі розглянемо і важливість кешування.. Нижче наведені результати обчислень даного алгоритму без кешування

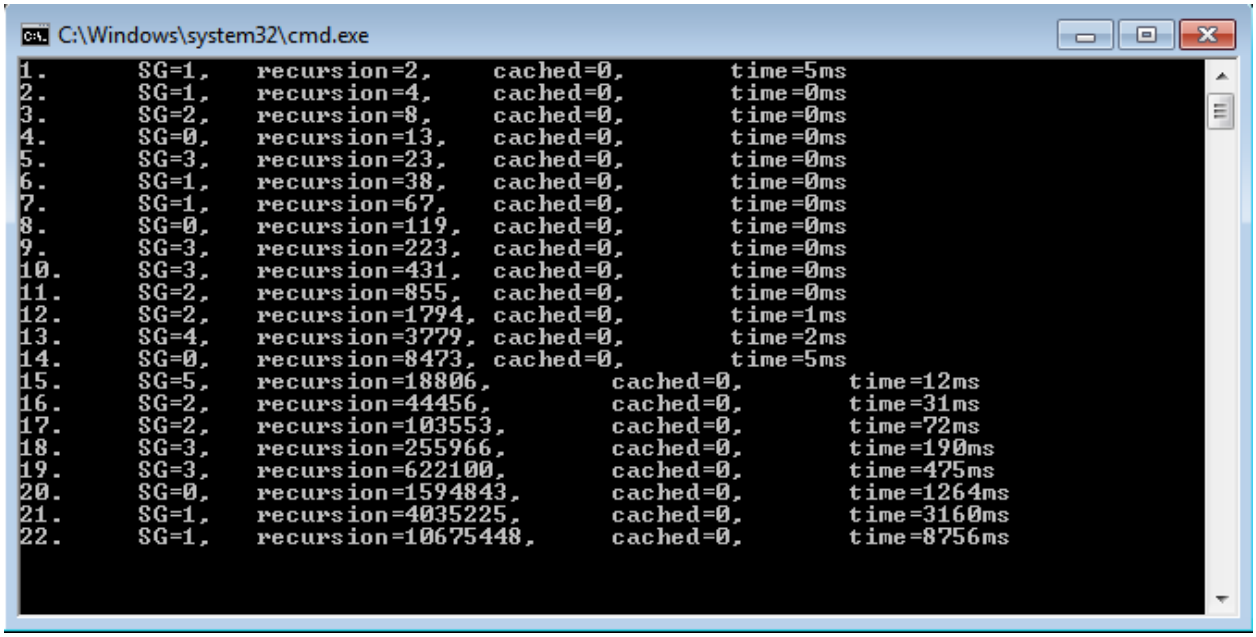


Рисунок 8. Результати обчислення гри Шахи Доусона без кешування

Глибина рекурсії досягає захмарним значень. І час обчислень росте в експоненційному порядку. Вже для значення n=22 результат рахувався біля 9 секунд, що є неприпустимо. Порівняємо з результатами з кешуванням.

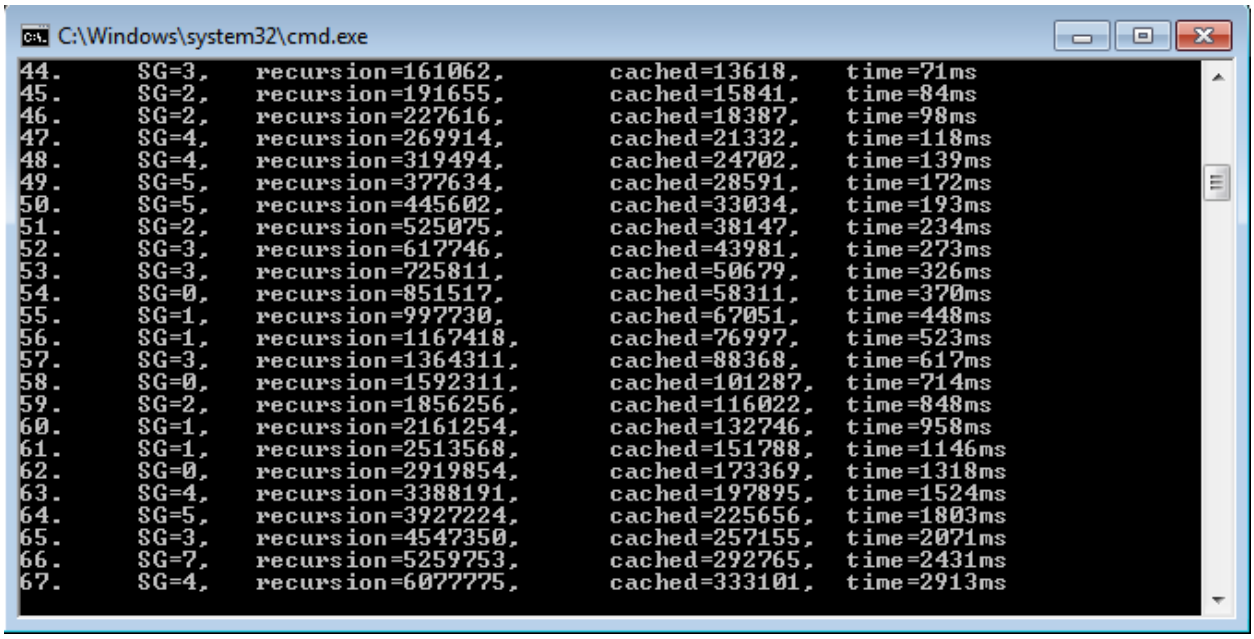


Рисунок 9. Результати обчислення гри Шахи Доусона з кешуванням

Як видно з рисунку 9, глибина рекурсії зменшилася більше ніж у 10 разів, і відповідно швидкість обчислень зросла. Тепер результати рахуються до 60их номерів без значної затримки. Але і це можна оптимізувати, якщо обчислення робити не прямолінійно, а враховувати властивість суми ігор.

Насправді, якщо трошки абстрагуватися від правил і подивитися на них з іншої сторони, можна побачити, що математична модель цієї гри повніст співпадає з математичною моделлю гри «хрестики-хрестики», що дозволє нам порахувати значення Шпрага-Гранді для цієї гри з глибиноб рекурсії до 3-4 вкладень і швидкість д лічені мілісекунди. Такий підхід ми можемо дозволити лише завдяки теоремі про суми ігор і їх еквівалентність нім-сумам цих ігор.

2.2.8 Гра Гризун (Chomp)

Тепер ми дійшли до останньої гри в цій роботі(тим не менше, цю гру в неявному вигляді придумали ще у 1952 році). Суть гри полягає в наступному: в нас є шоколадна плитка розміром $m \times n$. Гравці по черзі «відкушують» частину плитки, починаючив точці x,y і забриаючи з плитки всі клітки, що є над і справа від цієї точки. Клітинка (1,1)-отруєна. Хто її перший зїсть – той програє.

Розв’язок задачі зовсім не тривіальний. Більше того, в цій роботі ми його наводити не будемо. Ми розглядаємо цю гру, щоб продемонструвати неможливість застосування теорії Шпрага-Гранді до циклічних ігор.

Тим не менше, спробуємо проаналізувати цю гру. Маючи позицію (x,y) ми можемо в неї попасти з позицій

$$(x+i, y+j), i = -x+1 \dots m-x, j = -y+1 \dots n-y, i*j > 0$$

Тобто ми можемо попасти в позицію (x,y) з будь-якої позиції в полі, крім тих, що лежать на «південному заході». Попробуємо намалювати граф для перших декілька клітинок:

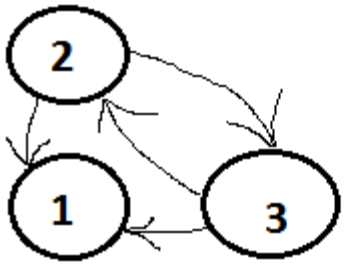


Рисунок 10. Граф для перших кроків гри «Гризун»

З рисунка 10 видно, що в графі присутні цикли, навтіь на найпростішому рівні. Це унеможливлює спосіб розв'язку Шпрага-Гранді для цієї задачі. Глянемо на наступний код

```
var set = new HashSet<uint>();

int y = key.Y;
int x = key.X;

for (int i = 1; i <= N; i++)
   for (int j = 1; j <= M; j++)
   {
      if ( i >= x && j >= y)
         break;
      set.Add(SGValue(new Coordinate(i, j)));
   }
```

При запуску його на виконання ми отримаємо StackOverflowException, оскільки функція буде викликати сама себе рекурсивно аж поки не закінчиться память в стеку для виділення памяті для локалних змінних і адреси повернення.

Таким чином, цю задачу не можна вирішити способом, розглянутим в даній роботі. Проте можна вказати підказку для оптимальної стратегії, а точніше легко довести, що гравець, котрий ходить перший завжди виграє. Це доводиться методом від супротивного. Припустім гравець номер 2 має виграшну стратегію проти будь-якого ходу гравця номер 1. Нехай тоді ходом гравця номер 1 буде найдальніша від початку координат клітинка. Тоді який би не був відповідний хід гравця номер 2, гравець 1 зможе його відтворити на початку гри.

2.3 Інтерфейс користувача

Дана програма призначена для людини, котра має хоча б базові знання з математики. Нагадаємо, що це консольна аплікація. Першим кроком запитується вибір гри, для якої ми хочемо передбачити результати. Після цього запитуються параметри цієї гри(номер клітинки з якої починати, розмір поля, та інші), після цього на екран виводиться результат – хто виграє в даній ситуації при оптимальній грі. Наступним кроком є повернення до кроку 1 – тобто вибір гри, або вихід з програми.

Для деяких ігор, в яких проводиться приведення гри до гри Нім, також можлива підказка щодо оптимального ходу в даній ситуації. Ця ситуація вираховується шляхом нім-додавання всіх змінних нім-рівняння крім однієї, і таким чиномми отримуємо значення, на яке потрібно поміняти змінну. Котру ми не брали до уваги. Якщо за логікою гри це не можливо, то шукаються решта допустими значень. Якщо таких не знайшлося – це означає що потрібно вдаватися до глибшого аналізу, або ж, що це P-позиція.

3. Висновки

В даній роботі ми розглянули теорію рівноправних ігор, їх класифікацію, багато прикладів, і що найголовніше – ми вивели алгоритм розв'язку великого підкласу таких ігор. Цей алгоритм ми міцно закріпили прикладами і зреалізованими програмами. Фактично ми отримали відповідь на запитання «хто ж виграє в цій грі в такій ситуації».

Також, ми знайшли шлях до пошуку відповіді на питання «який хід потрібно зробити щоб виграти цю гру». Оскільки відповідь на це питання сильно залежить від самої гри, то універсального способу не існує. Але тим не менше, ми знайшли раціональний підхід – розв'язувати нім-рівняння. Складність розв'язку таких рівнянь фактично являє O(n*n), де n-кількість змінних в рівнянні. Звісно це стосується нашого способу розв'язку. Цілком логічно, що при додаткових дослідженнях можна виявити й оптимальніші способи розв'язку таких рівнянь.

Іншим напіврозкритим питанням залишилося «а чи можна сказати якою буде позиція в будь-якому заданому стані за константний час?». Тобто чи є закономірність виведення NP-станів від ситуації в грі. Для деяких ігор таку закономірність ми знайшли, але для деяких ні.

Питання, якого ми зовсім не зачіпали, але яке варте уваги – «а що буде, якщо за подібними правилами будуть грати більше ніж двоє людей?». Тобто чи можна розширити теорію Шпрага-Гранді на випадок n гравців? І як тоді будуть відбуватися ходи? Хто залишиться у виграші? Як зміниться граф переходів станів, і які нові стани появляться? Як обирати оптимальну стратегію в такому випадку, і чи це взагалі можливо? Ці питання варті окремого дослідження і не є тривіальними, бо виходять за межі комбінаторних ігор.

Теорія, розглянута в даній роботі, дає основу багатьом іншим важливим галузям науки, зокрема економіки, соціології, масовому дослідженні поведінки та ін. Розвиток теорії ігор сприятиме покращенню прийняття рішень в системах з висо-

кою відповідальністю і великим ризиком, що значно покращить життя звичайних громадян.

Список літератури

1. E. R. Berlekamp, J. H. Conway and R. K. Guy (1982)Winning Ways for your mathematical plays, vols. 1 and 2, Academic Press, New York.
2. J. H. Conway (1976) On Numbers and Games, Academic Press, New York.
3. http://www.math.ucla.edu/~tom/Game_Theory/comb.pdf
4. http://e-maxx.ru/algo/sprague_grundy
5. http://jourdan.ens.fr/~laffargue/teaching/Incertain/Problemes/lectnotes.pdf
6. C. L. Bouton (1902) Nim, a game with a complete mathematical theory, Ann. Math. 3,35-39.
7. T. S. Ferguson (1998) Some chip transfer games, Theoretical Computer Science 191, 157-171.
8. http://web.mit.edu/sp.268/www/nim.pdf