Ворожцов Артём Викторович
Кандидат физико-математических наук,
преподаватель кафедры информатики
Московского физико-технического института (МФТИ),
тренер сборной команды МФТИ по программированию.

Теория игр: нимберы

Теория игр – одна из самых привлекательных теорий, и это неудивительно, вся наша жизнь – игра, и всем хотелось бы в этой игре выиграть. Математика и хитроумные алгоритмы иногда могут помочь в этом.

В данной статье рассмотрены нимберы – загадочные числа, которые позволяют во много раз облегчить поиск выигрышной стратегии в определённом классе игр.

Красивая теория, связанная с этими числами, довольна сложна. Здесь сделана попытка просто изложить основной результат этой теории, который поможет программистам конструировать выигрышные стратегии для простых логических игр.

Однажды мы уже касались это темы (см. №2 2005 г.). По многочисленным просьбам читателей мы снова возвращаемся к этой теме, на этот раз в контексте языка программирования Ruby, о котором много рассказывалось в последних номерах журнала.

Нимберы – это целые неотрицательные числа, которые можно поставить в соответствие игровым позициям в некоторых играх. Эти числа были придуманы в результате изучения игры Ним.

Игра Ним

Суть игры Ним. Дано несколько кучек камней. Игроки по очереди должны выбирать одну из кучек и брать из неё произвольное количество камней (возможно, все) больше 0. Выигрывает тот, кто взял последний камень.

Рассмотрим пример. Будем обозначать набор кучек набором чисел, соответствующих количествам камней в кучках. Например, позиция {5,3,2} – это позиция, когда есть три кучки: в первой – 5, во второй – 3, а в третьей – 2.

Из позиции {3,2,1} первый игрок может попасть в позиции {2,1} (взяв все камни из первой кучки), {2,1,1} (взяв два камня из первой кучки),

{2,2,1} (взяв один камень из первой кучки), {3,1} (взяв все камни из второй кучки), {3,1,1}, {3,2}. Указывая позиции, мы будем упорядочивать количества по убыванию. Действительно, нам неважно, в каком порядке идут кучки, поэтому, чтобы не рассматривать одну и ту же позицию дважды, удобно упорядочивать числа в наборе. Кроме того, мы не будем рассматривать кучки, в которых осталось ноль камней.

Предположим, что первый сходил в позицию {2,2,1}. Тогда второй мог сходить в позицию {2,2}, взяв один камень из последней кучки. Затем первый мог сходить в позицию {2,1}, второй – в позицию {1,1}, первый – в позицию {1}, и, наконец, второй берёт один камень из последней кучки и ходит в позицию {}, которая является проигрышной. Тот игрок, кому она достаётся и из которой ему нужно ходить, проигрывает.

Задача. Рассмотрите все возможные варианты развития игры Ним и покажите, что как бы ни ходил первый игрок, второй игрок всегда сможет выиграть.

На первый взгляд, задача кажется сложной, так как кажется, что необходимо рассмотреть огромное количество вариантов. Но в действительности всё гораздо проще.

Можно нарисовать карту всех позиций (граф игры). Позиции, из которых можно сходить в матовую, отметить как выигрышные. Позиции, из которых можно сходить только в выигрышные, отметить как проигрышные. Затем позиции, из которых можно сходить хотя бы в одну позицию, которая является проигрышной, отметить как выигрышные и т.д. Шаги определения статуса позиций показаны на рис. 1.

Есть интересная теорема, позволяющая определять статус позиции игры Ним.

Теорема 1 (о нимбере игры Ним).

Чтобы определить статус позиции игры Ним, необходимо:

1) количество камней перевести в двоичную систему счисления;

2) осуществить побитовое сложение чисел по модулю 2 (то есть без переноса на следующий разряд); эта операция в программировании называется побитовый XOR (eXclusive OR) и обычно обозначается как ^. Результат побитового XOR'а размеров кучек называется *нимбером позиции игры Ним* (*nimber* – гибрид английских слов *nim* и *number*);

3) если результат побитового сложения равен 0, то позиция проигрышная, иначе – выигрышная.

Есть альтернативная формулировка данной теоремы: представим размер каждой кучки в виде суммы неповторяющихся степеней двойки (это разложение существует и единственное). Рассмотрим все получившиеся степени двойки в совокупности.

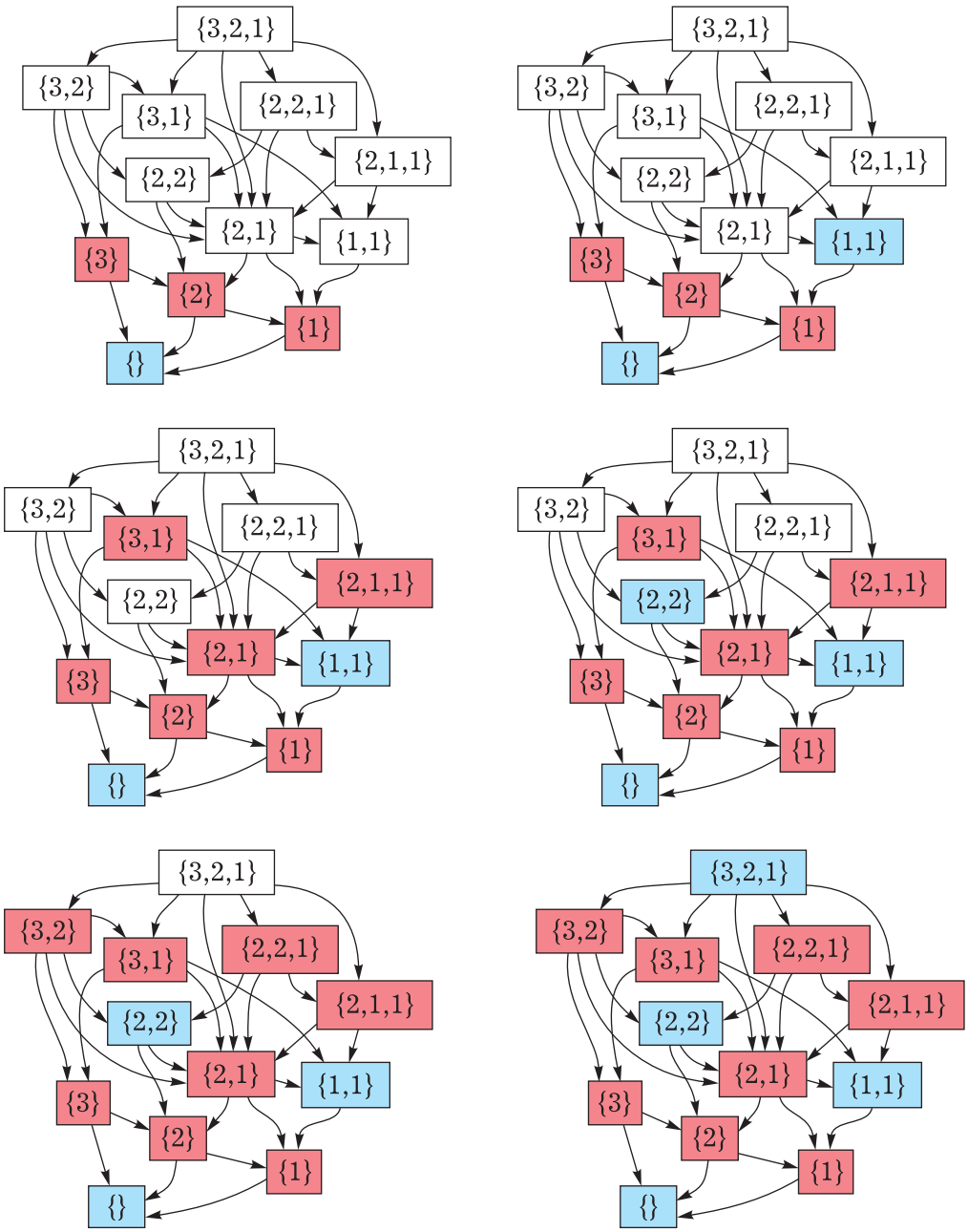

Рис. 1. Граф игры Ним – множество позиций, в которые можно попасть из позиции {3,2,1}, соединённые стрелками, которые отображают возможные ходы. Показаны шаги пошагового назначения позициям статуса. Синие позиции являются проигрышными (в них при правильной игре второго игрока проигрывает тот, кто ходит первый), красные позиции – выигрышные (в этих позициях выигрывает первый игрок; он может сходить в проигрышную позицию).

Если каждая степень в этой совокупности представлена чётным числом раз, то в игре выигрывает второй (если, конечно, он будет правильно ходить).

Напишем программу для вычисления нимбера позиции игры Ним.

Код на языке Ruby с примерами

```ruby
# Метод получает массив a целых чисел
# и возвращает нимбер соответствующей
# позиции  игры Ним
def nimber(a)
  a.inject(0) {|res,x| res ^ x}
end

# Нимберы проигрышных позиций равны 0
# и следующие три строки напечатают
# три нуля
puts nimber([3,2,1])
puts nimber([2,2])
puts nimber([1,1])

# Нимберы проигрышных позиций не равны 0
puts nimber([3,2]) # => 1
puts nimber([3,1]) # => 2
puts nimber([2])   # => 2
```

Код на языке Си с примерами

```c
#include <stdio.h>
int nimber(int *a, int n) {
  int i, res = 0;
  for(i = 0; i < n ; i++) {
    res ^= a[i]
  }
  return res;
}
int main() {
  int a[3] = {3,2,1};
  int b[2] = {2,2};
  int c[2] = {3,2};
  printf("%d\n%d\n%d\n",
    nimber(a,3),
    nimber(b,2),
    nimber(c,2)
  );
}
```

Теорема о нимбере суммы игр

Интересно, что теорему 1 можно обобщить, а именно, вычислять нимберы и для других игр, и при этом пункт (3) теоремы остаётся в силе. Для формулировки обобщающей теоремы необходимо дать определение «суммы игры», а также понятие конечности и нейтральности игры.

Определение 1. Игра называется *конечной*, если для любой её позиции можно указать ограничение на число ходов, которое могут проделать игроки до момента, когда один из них выиграет.

Определение 2. Игра называется *нейтральной*, если множества возможных ходов для игроков не отличаются, то есть из одной и той же позиции игроки могут сделать одно и то же множество ходов.

Определение 3. Пусть даны две игры (две игровые позиции) A и B, в которых условие проигрыша звучит так: проигрывает тот, кто не может сделать ход. Тогда *сумма игр* $C = A + B$ по определению есть игра, которая ведётся «на двух досках одновременно»: на одной доске игроки играют в игру A, а на другой – в игру B. При этом ход заключается в том, что игрок сначала выбирает доску, на которой он хочет сделать ход, а потом делает там ход. При этом на второй доске ситуация не меняется, и он на ней как бы пропускает ход. В игре C проигрывает тот игрок, который не может сделать ход ни на одной из двух досок.

Определение 4. *Нимбер позиции произвольной конечной нейтральной игры* A (некоторой фиксированной позиции в этой игре) равен минимальному неотрицательному целому числу, которое отсутствует среди

множества чисел M, которое равно множеству нимберов позиций, в которые можно сходить из позиции A.

И эту цепочку определений завершим главной теоремой, которая, собственно, и является той жемчужиной, которую уже можно использовать на практике.

Теорема 2 (о нимбере суммы игр). При сложении конечных нейтральных игр (позиций игр) получается конечная нейтральная игра и нимбер этой игры равен побитовому XOR'у нимберов складываемых игр. Игры, для которых нимберы равны нулю, являются проигрышными, а именно, у второго игрока есть выигрышная стратегия, в остальных играх есть выигрышная стратегия у первого игрока:

$$nimber(A+B) = nimber(A) \wedge nimber(B)$$

$nimber(A) = 0 <=>$ в игре A есть выигрышная стратегия у второго

Рекурсивное определение 4 удобно записать как метод на языке Ruby:

```
def ni(n)
  i=0
  moves(n).map {|x| ni(x)}.uniq.sort.each do |x|
    return i if x != i
    i+=1
  end
  return i
end
```

В этом коде предполагается, что метод **moves(p)** возвращает множество позиций, в которые можно попасть из позиции p. Затем метод **map {|x| ni(x)}** превращает массив позиций в массив их нимберов, затем метод **uniq** удаляет из этого массива повторения, и, наконец, результат сортируется методом **sort**. В результате получается массив, который состоит из неотрицательных целых чисел, и необходимо найти минимальный индекс, на котором этот массив отличается от массива **[0,1,2,3,4,…]**.

В случае «однокучковых» позиций игры Ним всё очень просто: позиция – это просто число камней, а метод **moves(n)** должен возвращать массив **[0,1,…, n-1]**:

```
def moves(n)
  # три точки в определении диапазона обозначают,
  # что последнее число не включается;
  # метод to_a диапазон превращает в массив
  (0...n).to_a
end
```

В результате имеем очевидную вещь: **(0..10).map {|p| ni(p)}** возвращает массив **[0,1,2,3,4,5,6,7,8,9,10]**. Нимбер позиции из одной кучки равен количеству камней в ней.

NB! Игра Ним с несколькими кучками является суммой «однокучковых» игр и теорема 1 является частным случаем теоремы 2.

Модифицированная игра Ним

Для других игр результат будет не таким очевидным.

Давайте напишем функцию **moves(p)** для несколько модифицированной игры Ним: за ход можно взять из кучи не более половины камней:

```
def moves(n)
  (n+1/2...n).to_a
end
```

Тогда результат будет более интересным. Но прежде, чем к нему обратиться, давайте напишем метод вычисления нимбера общего назначения (годный для любой игры). В этом методе мы сначала узнаем, какого класса есть аргумент. Если он имеет класс Array, то будем считать, что необходимо посчитать нимбер от суммы игр как XOR всех их нимберов. Иначе же будем считать, что передана одна позиция, и вычислим нимбер, следуя рекурсивному определению 4. Кроме того, следуя идеологии Ruby (Ruby way), сделаем передачу метода **moves** как *ассоциированного блока*. В итоге получим:

```
def nimber(p,&moves)
  if p.instance_of? Array
    return p.inject(0){|res,x| res ^ nimber(x,&moves)}
  else
    i = 0
    moves[p].map {|x| nimber(x,&moves)}.uniq.sort.each do |x|
      break if i!=x
      i+=1
    end
    return i
  end
end
```

В результате код

```
p (1..15).map {|n|
  nimber(n) {|x| ((x+1)/2...x)}
};
```

вернёт массив **[0,1,0,2,1,3,0,4,2,5,1,6,3,7,0]**.

Блок **moves** является аргументом метода **nimber**. Символ & перед ним означает, что это не обычный объект, а блок. Блок – это определённая «по ходу дела» функция (анонимная функция). Вызов этой функции с аргументом **p** осуществляется с помощью квадратных скобок: **moves[p]**. При вызове блока амперсанд перед его именем писать уже не нужно.

Для того, чтобы проводить дальнейшие вычисления, полезно сделать *кэширование уже вычисленных значений*. Для этого мы заведём глобальную переменную **$c** класса Hash, в которой будем хранить пары {позиция => вычисленный для неё нимбер}. Заметьте, что имена глобальных переменных в языке Ruby начинаются на доллар. Для реализации кэширования добавляется одна строчка **return $c[p] if $c.has_key?(p)** в начале метода, и перед возвращением вычисленного значения необходимо добавить его занесение в глобальный хэш **$c: return ($c[p]=i)**. В итоге получаем следующий код:

```
$c={}
def nimber(p,&moves)
  if p.instance_of? Array
    return p.inject(0){|res,x| res ^ nimber(x,&moves)}
  else
    return $c[p] if $c.has_key?(p)
    i=0
    moves[p].map {|x| nimber(x,&moves)}.uniq.sort.each do |x|
      break if i!=x
      i+=1
    end
    return ($c[p]=i)
  end
end
```

Данный код является достаточно простым и при этом универсальным. С его помощью можно вычислить нимбер позиции произвольной игры. Необходимо просто вызвать этот метод для позиции и задать блок, принимающий позицию и возвращающий массив (или диапазон) позиций, в которые можно попасть из этой позиции.

Например, чтобы найти нимбер позиции {132,73,23}, необходимо осуществить следующий вызов метода **nimber**:

```
p nimber([132,73,23]) {|x| ((x+1)/2...x)}
```

Чтобы найти нимберы 150 «одно-кучковых» позиций модифицированной игры Ним, достаточно написать следующий код:

```
p (1..150).map {|n|
  nimber(n) {|x| ((x+1)/2...x)}
}
```

Результат равен очень интересной последовательности:

```
  0, 1, 0, 2, 1, 3, 0, 4, 2, 5, 1, 6, 3, 7, 0, 8, 4, 9, 2, 10,
5, 11, 1, 12, 6, 13, 3, 14, 7, 15, 0, 16, 8, 17, 4, 18, 9, 19,
2, 20, 10, 21, 5, 22, 11, 23, 1, 24, 12, 25, 6, 26, 13, 27, 3,
28, 14, 29, 7, 30, 15, 31, 0, 32, 16, 33, 8, 34, 17, 35, 4, 36,
18, 37, 9, 38, 19, 39, 2, 40, 20, 41, 10, 42, 21, 43, 5, 44, 22,
45, 11, 46, 23, 47, 1, 48, 24, 49, 12, 50, 25, 51, 6, 52, 26,
53, 13, 54, 27, 55, 3, 56, 28, 57, 14, 58, 29, 59,…
```

Если выкинуть из неё первый элемент, а затем взять каждый второй, то получим снова её же. Чётные элементы этой последовательности являются последовательными натуральными числами.

Задача. Докажите это утверждение. В помощь может прийти двоичное представление результатов. Давайте выведем двоичную запись размера кучки и соответствующего ей нимбера:

```
(1..150).each {|n|
  res = nimber(n) {|x| ((x+1)/2...x)}
  puts "#{n.to_s(2)}: #{res.to_s(2)}"
}
```

Результат выглядит следующим образом:

```
1: 0
10: 1
11: 0
100: 10
101: 1
110: 11
111: 0
1000: 100
1001: 10
1010: 101
1011: 1
1100: 110
....(выкинут кусок)
11110: 1111
11111: 0
....(выкинут кусок)
10001111: 100
10010000: 1001000
10010001: 100100
10010010: 1001001
10010011: 10010
10010100: 1001010
10010101: 100101
10010110: 1001011
```

Задача. Угадайте закономерность в двоичном виде нимберов модифицированной игры Ним. Как можно получить двоичную запись **nimber(n)** по двоичной записи числа n?

Задача. Напишите программу, которая вычисляет нимбер следующей игры: на столе стоит ряд из n фишек. За ход игроку разрешается снять со стола любую фишку или любые две рядом стоящие фишки (фишки, между которыми есть промежуток, не считаются соседними). Выигрывает тот, кто возьмёт последнюю фишку.

Сложный материал

Абстрактное определение понятия игры

Давайте попробуем ввести абстрактное математическое понятие игры.

Игра имеет множество игровых ситуаций. Правила игры определяются отображением m, которое каждой позиции p ставит в соответствие множество позиций $m(p)$, в которые можно попасть из p за один ход. Игру можно представлять как ориентированный граф, в котором вершины есть игровые позиции, а рёбра отображают возможные ходы. В некоторых играх возможные ходы для игроков отличаются. Тогда рёбра графа раскрашиваются в два цвета, например, синий и красный: по синим рёбрам может ходить только «синий» игрок, а по красным рёбрам – «красный» игрок.

Игры, в которых синие и красные рёбра дублируют друг друга, называются нейтральными, другими словами, игра нейтральна, если из любой позиции множество ходов, которые может сделать один игрок, совпадает с множеством ходов, которые может сделать другой игрок.

Игра «шахматы» не является нейтральной, так как игрок не может двигать чужие фигуры. В шахматах ни одна пара красного и синего ребра не повторяет друг друга. А игра Ним

нейтральная.

Игра называется конечной, если для любой позиции этой игры можно ограничить количество ходов, которое могут сделать игроки до момента, когда кто-либо из них выиграет, другими словами, игра конечна, если любой путь, начинающийся из какой-либо вершины, имеет конечную длину. Если граф игры конечен, то это просто означает, что в графе нет циклов.

Определение нимбера $n(p)$ от позиции p любой конечной нейтральной игры выглядит следующим образом:

$$n(p) = \min\left(N_0 \setminus \{n(q) \mid q \in m(p)\}\right),$$

где $N_0 = \{0,1,2,...\}$.

Конечность игры необходима для того, чтобы это рекурсивное определение было корректно.

Калейдоскоп Калейдоскоп Калейдоскоп

Поддерживать в течение суток практически постоянную температуру в помещениях удаётся в здании, построенном в Германии.

Достигнуто это благодаря введению в штукатурку стен пластиковых микрокапсул, наполненных легкоплавким веществом парафином. Когда температура поднимается выше 24° С, парафин начинает плавиться, поглощая при этом процессе теплоту. Если же температура понижается, то он, остывая и затвердевая, выделяет теплоту. К тому же штукатурка с парафином обладает хорошими теплоизолирующими свойствами: её 2-сантиметровый слой эквивалентен в этом отношении 20 см кирпича.