

Алексей Словеснов

email slovesnov@yandex.ru

сайт <http://bulls-cows.sourceforge.net>

Поиск оптимальных алгоритмов в игре быки-коровы.

Предисловие.

В статье рассматривается оптимизация игры быки-коровы по двум критериям. Первый - минимизация числа номеров, которые компьютер угадывает ровно за семь ходов, при этом остальные номера должны угадываться не более чем за шесть ходов. Второй - минимизация среднего числа ходов, которое тратится на отгадывание произвольного номера (минимизация средней длины игры). В процессе поиска были разработаны промежуточные алгоритмы, для которых построены деревья и написано веб приложение для игры компьютера с человеком.

В данный момент осуществляется поиск серверов для завершения расчетов и нахождения точной оценки. Если вы хотите помочь проекту - свяжитесь с автором статьи.

1 ноября 2011

вторая редакция

адрес последней версии статьи
<http://slovesnov.narod.ru/articles/bullcow.pdf>

Содержание

Введение.	3
1 Теория.	4
1.1 Обозначения.	4
1.2 Преобразования.	5
1.3 Классы эквивалентности.	6
1.4 Множество первых ходов.	7
1.5 Множество вторых ходов.	7
1.6 Множество третьих ходов.	8
1.7 Оценки подмножеств.	9
1.8 Минимально возможная оценка.	9
1.9 Быстрая оценка.	10
1.10 Отсечения для ходов 4-6.	11
1.10.1 Отсутствующие цифры.	11
1.10.2 Неназванные цифры.	11
1.10.3 Ходы разбивающие множество номеров на одну группу.	12
1.11 На несколько ходов получены одинаковые ответы.	12
2 Алгоритмы.	13
2.1 Точный алгоритм.	13
2.1.1 Алгоритм последнего слоя.	13
2.2 Эвристические алгоритмы.	14
2.2.1 Дробящий алгоритм.	14
2.2.2 Алгоритм Лармауса.	14
2.3 Ускорения алгоритмов.	15
2.4 Обозначения алгоритмов.	15
2.5 Расширение для подсчета шестиходов, пятиходов итд.	15
3 Компоненты алгоритма.	16
3.1 Альфа-бета отсечения.	16
3.2 Массивы второго, третьего ходов и фиксированный первый ход.	16
3.3 Упорядочивание ходов 2 и 3.	16
3.4 Упорядочивание остальных ходов.	16
3.5 Таблица ответов.	16
3.6 Механизм задач.	17
4 Компоненты программы.	17
4.1 Консольное приложение executor.	18
4.2 Сервис windows.	18
4.3 Приложение под windows - bcw.	18
4.4 Консольное приложение shutdown.	18
4.5 Приложение tree на javascript.	18

5	Результаты минимизации семиходовок.	19
5.1	Алгоритмы crush35 и crush45.	19
5.2	Переход от одного алгоритма к другому.	19
5.3	Алгоритм crush5.	21
5.3.1	Расчет для лучших ходов (шаги 1-5).	21
5.3.2	Построение таблицы (шаг 6).	22
5.3.3	Лучший ход после (0123,0.2) (шаги 7-8).	22
5.3.4	Лучший ход после (0123,0.1) (шаг 9).	23
5.3.5	Проверка номеров с цифрами 0-3 (шаг 10).	23
5.3.6	Результаты алгоритма crush5.	24
5.4	Точный алгоритм.	24
5.5	Сравнительная таблица результатов алгоритмов.	24
6	Минимизация числа шестиходовок, пятиходовок итд.	24
7	Результаты минимизации средней длины игры.	25
8	Построение дерева ходов.	25
8.1	Реализация алгоритма на c++.	26
8.2	Реализация алгоритма на javascript.	26
9	Время расчетов.	27
9.1	Алгоритмы crush35 и crush45.	27
9.2	Алгоритм crush5.	28
9.3	Алгоритмы avg35 и avg45.	28
9.4	Алгоритм avg5.	29
10	История редакций.	29
	Список литературы.	29

Введение.

Оптимизация алгоритмов для игры быки-коровы традиционно ведется в двух направлениях.

- Первое направление. Минимизация среднего числа ходов, которое тратится на отгадывание произвольно загаданного номера. Эта задача решена (см. [1] и [2]), среднее число ходов равно $26274/5040=5.2131$. Достаточно лишь найти алгоритм с таким средним числом ходов.
- Второе направление. Известно, что нельзя придумать алгоритм, в котором любой номер угадывался бы за шесть и менее ходов, в то же время известно, что существуют алгоритмы, которые угадывают все номера не более чем за семь ходов. Исходя из этого, получаем второе направление оптимизации - минимизировать число номеров, которое угадывается ровно за семь ходов.

Нахождение минимальной средней длины игры или минимального числа номеров, угадываемых ровно за семь ходов, является чрезвычайно трудоемкой задачей. Попробуем грубо оценить число вариантов в алгоритме полного перебора. Очевидно, что в качестве первого хода, достаточно рассмотреть номер 0123. Это несколько не ограничивает общности. Ходы во второго по шестой имеют по 5039 вариантов (ход 0123 не используется). После каждого из ходов можно получить максимум 14 ответов, по одному из которых (четыре быка и ноль коров) дальнейший поиск не ведется. Будем считать, что с помощью алгоритма, можно уменьшить среднее число перебираемых вариантов в четыре раза и среднее число ответов также в четыре раза. Получаем оценку числа вариантов для полного перебора $5039^5 \times (\frac{13}{16})^6 \approx 9.3 \times 10^{17}$. Это просто астрономическая цифра даже для современных компьютеров.

Статья разбита на несколько частей. В первой вводятся обозначения и дается математическая теория. Во второй описываются алгоритмы. В третьей рассматриваются основные узлы алгоритма перебора. В четвертой кратко описаны компоненты программы. В пятой показаны результаты работы алгоритмов минимизации числа номеров, угадываемых ровно за семь ходов. В шестой алгоритм будет расширен для минимизации не только числа номеров угадываемых ровно за семь ходов, но и за шесть, пять и так далее ходов. В седьмой части описываются результаты минимизации средней длины игры. В восьмой описывается построение деревьев ходов, которые будут использоваться для веб приложения, где компьютер будет отгадывать номера, задуманные человеком. В девятой части приводятся данные о времени расчетов.

1 Теория.

1.1 Обозначения.

Ответ на номер обозначается - „число быков.число коров“. Например, 2.1 означает, что получен ответ два быка и одна корова.

Номер t с ответом r называется ходом, обозначается - (t, r) . Иногда, ход будет писаться без скобок и запятой - 5678 0.1.

Множество всех номеров обозначаем за $\Omega = (0123, 0124 \dots 9876)$.

Множество всех ответов обозначаем за $R = (0.0, 0.1 \dots 4.0)$.

Последовательность ходов обозначим за $(t_1, r_1) \dots (t_n, r_n)$.

Номера угадываемые ровно за k попыток будут называться „ k -ходовками“.

Минимальное число номеров, среди всех алгоритмов, угадываемых ровно за k попыток, после последовательности ходов $(t_1, r_1) \dots (t_n, r_n)$ обозначим за $\Phi_k[(t_1, r_1) \dots (t_n, r_n)]$.

Минимальную длину игры, среди всех алгоритмов, после последовательности ходов $(t_1, r_1) \dots (t_n, r_n)$ обозначим за $\Delta[(t_1, r_1) \dots (t_n, r_n)]$. Предположим, что после последовательности ходов осталось m номеров. Для того, чтобы работать с целыми числами будем использовать аналогичную функцию равную сумме числа ходов для отгадывания всех оставшихся номеров. $\Lambda[(t_1, r_1) \dots (t_n, r_n)] = m \times \Delta[(t_1, r_1) \dots (t_n, r_n)]$.

Рассмотрим ходы $(t_1, r_1) \dots (t_n, r_n)$ и номер p . Сумму числа номеров угадываемых ровно за k попыток, после ходов $(t_1, r_1) \dots (t_n, r_n)$ и следующего хода p с любым возможным ответом обозначим $\Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, *)]$.

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, *)] = \sum_{r \in R} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, r)] \quad (1)$$

Аналогично,

$$\Lambda[(t_1, r_1) \dots (t_n, r_n)(p, *)] = \sum_{r \in R} \Lambda[(t_1, r_1) \dots (t_n, r_n)(p, r)] \quad (2)$$

Каждый номер состоит из четырех цифр, каждая из них занимает позицию от 1 до 4. Первая цифра имеет позицию один, вторая два и так далее. Каждая цифра может быть от 0 до 9. Обозначим множество всевозможных перестановок цифр от 0 до 9 и позиций от 1 до 4 за Ψ , а элементы этого множества $\psi \in \Psi$.

Результат действия преобразования ψ на номер t , будем обозначать $\psi(t)$.

Результат действия преобразования ψ на множество номеров $T = \{t_1 \dots t_n\}$ будем обозначать множество номеров $\{\psi(t_1) \dots \psi(t_n)\}$. $\psi(T) = \{\psi(t_1) \dots \psi(t_n)\}$.

Обозначим мощность множества S (число элементов в нем) через $|S|$. Например, $|\Psi| = 10! \times 4! = 87\,091\,200$.

Пустое множество - \emptyset .

Цель данной статьи найти $\Phi_7[\emptyset]$ и $\Lambda[\emptyset]$.

1.2 Преобразования.

Можно делать любое переобозначение цифр и позиций, которое по сути ничего не меняет, поэтому справедливо следующее утверждение:

Лемма 1 Пусть есть последовательность ходов $(t_1, r_1) \dots (t_n, r_n)$, и преобразование ψ , тогда

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)] = \Phi_k[(\psi(t_1), r_1) \dots (\psi(t_n), r_n)]$$

$$\Lambda(t_1, r_1) \dots (t_n, r_n) = \Lambda[(\psi(t_1), r_1) \dots (\psi(t_n), r_n)]$$

Лемма 2 Для любых двух номеров p_1 и p_2 существует преобразование ψ такое что $\psi(p_1) = p_2$ и $\psi(p_2) = p_1$.

Доказательство. Написав, компьютерную программу перебора всех номеров $p_1 \in \Omega$ и $p_2 \in \Omega$ и всех преобразований $\psi \in \Psi$, можно убедиться, что утверждение верно.

Пример. Рассмотрим два номера 0123 и 4051. Придумайте преобразование ψ , такое что $\psi(0123) = 4051$, $\psi(4051) = 0123$.

Решение. Рассмотрим преобразование ψ_1 , которое меняет цифры $0 \leftrightarrow 1$, $2 \leftrightarrow 5$, $3 \leftrightarrow 4$. Номер 0123 переходит в номер $\psi_1(0123) = 1054$, а номер 4051 переходит в $\psi_1(4051) = 3120$. Теперь рассмотрим преобразование позиций ψ_2 $1 \leftrightarrow 4$, цифры в первой и четвертой позиции меняются местами. Применив преобразование ψ_2 , получим $\psi_2(\psi_1(0123)) = \psi_2(1054) = 4051$. Аналогично $\psi_2(\psi_1(4051)) = \psi_2(3120) = 0123$. Таким образом композиция $\psi = \psi_2 \circ \psi_1$ дает нужный результат.

Лемма 3 Для любых двух номеров p_1 и p_2 и любых двух ответов r_1 и r_2

$$\Phi_k[(p_1, r_1)(p_2, r_2)] = \Phi_k[(p_1, r_2)(p_2, r_1)]$$

$$\Phi_k[(p_1, r_1)(p_2, *)] = \Phi_k[(p_2, r_1)(p_1, *)]$$

Доказательство. Из леммы 2 следует, что существует ψ , такое что $\psi(p_1) = p_2$, $\psi(p_2) = p_1$, тогда, используя лемму 1, получаем что:

$$\Phi_k[(p_1, r_1)(p_2, r_2)] = \Phi_k[(\psi(p_1), r_1)(\psi(p_2), r_2)] = \Phi_k[(p_2, r_1)(p_1, r_2)]$$

Так как порядок ходов неважен, то ходы 1 и 2 можно переставить местами, поэтому

$$\Phi_k[(p_2, r_1)(p_1, r_2)] = \Phi_k[(p_1, r_2)(p_2, r_1)]$$

Первое доказано. Аналогично получаем, что

$$\Phi_k[(p_1, r_1)(p_2, *)] = \sum_r \Phi_k[(p_1, r_1)(p_2, r)] = \sum_r \Phi_k[(p_2, r_1)(p_1, r)] = \Phi_k[(p_2, r_1)(p_1, *)]$$

Утверждение доказано.

Примечание. Тоже самое верно и для функции Λ .

Лемма 4 Для любого номера p , любого ответа r .

$$\Phi_k[(0123, r)] = \Phi_k[(p, r)]$$

$$\Phi_k[(0123, *)] = \Phi_k[(p, *)]$$

Доказательство. Рассмотрим номер 0123 и любой номер p . Тогда из леммы 2 следует, что есть преобразование ψ , такое что $\psi(0123) = p$. Если взять в лемме 1 в качестве последовательности ход (0123, r) с неким ответом r , тогда получим, что $\Phi_k[(0123, r)] = \Phi_k[(\psi(0123), r)] = \Phi_k[(p, r)]$, также ясно, что $\Phi_k[(0123, *)] = \Phi_k[(p, *)]$ так как

$$\Phi_k[(0123, *)] = \sum_r \Phi_k[(0123, r)] = \sum_r \Phi_k[(\psi(0123), r)] = \sum_r \Phi_k[(p, r)] = \Phi_k[(p, *)]$$

Утверждение доказано.

Примечание. Тоже самое верно и для функции Λ .

1.3 Классы эквивалентности.

Назовем два номера p и q эквивалентными относительно последовательности ходов $(t_1, r_1) \dots (t_n, r_n)$ ($p \sim q$) если $\exists \psi : \psi(p) = q$ и $\psi(t_i) = t_i, 1 \leq i \leq n$.

Лемма 5 Отношение $p \sim q$ является отношением эквивалентности.

Доказательство.

1. Докажем, что $a \sim a$ (рефлексивность).

Очевидно, достаточно взять тождественное преобразование.

2. Докажем, что если $a \sim b$, то $b \sim a$ (симметричность).

Так как $a \sim b$, то $\exists \psi : \psi(a) = b, \psi(t_i) = t_i$. У каждого преобразования есть обратное преобразование $\psi^{-1} : \psi^{-1}(b) = a, \psi^{-1}(t_i) = t_i$. Тем самым, симметричность доказана.

3. Докажем, что если $a \sim b$ и $b \sim c$, то $a \sim c$ (транзитивность).

Так как $a \sim b$, то $\exists \psi_1 : \psi_1(a) = b, \psi_1(t_i) = t_i$. Из того что $b \sim c$, следует что $\exists \psi_2 : \psi_2(b) = c, \psi_2(t_i) = t_i$. Очевидно, что $\psi_2 \circ \psi_1(a) = \psi_2(b) = c$ и $\psi_2 \circ \psi_1(t_i) = \psi_2(t_i) = t_i$. Тем самым, транзитивность доказана.

Лемма 6 Если два номера p и q принадлежат одному классу эквивалентности, по последовательности $(t_1, r_1) \dots (t_n, r_n)$ то $\forall k, \forall r$

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, r)] = \Phi_k[(t_1, r_1) \dots (t_n, r_n)(q, r)]$$

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, *)] = \Phi_k[(t_1, r_1) \dots (t_n, r_n)(q, *)]$$

Доказательство. Если $p \sim q$, то $\exists \psi : \psi(p) = q, \psi(t_i) = t_i$ Из леммы 1, получаем что

$$\begin{aligned}\Phi_k[(t_1, r_1) \cdots (t_n, r_n)(p, r)] &= \Phi_k[(\psi(t_1), r_1) \cdots (\psi(t_n), r_n)(\psi(p), r)] = \\ &= \Phi_k[(t_1, r_1) \cdots (t_n, r_n)(q, r)]\end{aligned}$$

Аналогично,

$$\begin{aligned}\Phi_k[(t_1, r_1) \cdots (t_n, r_n)(p, *)] &= \sum_r \Phi_k[(t_1, r_1) \cdots (t_n, r_n)(p, r)] = \\ = \sum_r \Phi_k[(\psi(t_1), r_1) \cdots (\psi(t_n), r_n)(\psi(p), r)] &= \sum_r \Phi_k[(t_1, r_1) \cdots (t_n, r_n)(q, r)] = \\ &= \Phi_k[(t_1, r_1) \cdots (t_n, r_n)(q, *)]\end{aligned}$$

Утверждение доказано.

Примечание. Тоже самое верно и для функции Λ .

1.4 Множество первых ходов.

Лемма 7 В качестве первого хода достаточно рассмотреть только номер 0123, остальные можно отбросить.

$$\Phi_k[\emptyset] = \Phi_k[(0123, *)]$$

Доказательство. Чтобы найти $\Phi_k[\emptyset]$ надо перебрать в качестве первого хода все возможные ходы из Ω и взять минимальную оценку

$$\Phi_k[\emptyset] = \min_{p \in \Omega} \Phi_k[(p, *)]$$

Из леммы 4 следует, что для любого номера p $\Phi_k[(p, *)] = \Phi_k[(0123, *)]$, поэтому

$$\Phi_k[\emptyset] = \min_{p \in \Omega} \Phi_k[(p, *)] = \Phi_k[(0123, *)]$$

Утверждение доказано.

Примечание. Тоже самое верно и для функции Λ .

1.5 Множество вторых ходов.

Лемма 8 Если был сделан первый ход 0123, то в качестве второго хода достаточно рассмотреть номера из множества S_2 .

$$\begin{aligned}S_2 &= (0124, 0132, 0134, 0145, 0214, 0231, 0234, 0245, 0456, \\ &1032, 1034, 1045, 1204, 1230, 1234, 1245, 1435, 1456, 4567)\end{aligned}$$

или

$$\Phi_k[(0123, r)] = \min_{p \in S_2} \Phi_k[(0123, r)(p, *)]$$

Доказательство. Рассмотрим два номера p и q . Если есть преобразование ψ такое что, $\psi(0123) = 0123, \psi(p) = q$, тогда

$$\Phi_k[(0123, r)(p, *)] = \Phi_k[(\psi(0123), r)(\psi(p), *)] = \Phi_k[(0123, r)(q, *)]$$

Аналогично, если существует преобразование ϕ такое что, $\phi(0123) = q, \phi(p) = 0123$, тогда

$$\begin{aligned} \Phi_k[(0123, r)(p, *)] &= \Phi_k[(\phi(0123), r)(\phi(p), *)] = \\ &= \Phi_k[(q, r)(0123, *)] = \sum_{e \in R} \Phi_k[(q, r)(0123, e)] \end{aligned}$$

используя лемму 3, получаем, что

$$\sum_{e \in R} \Phi_k[(q, r)(0123, e)] = \sum_{e \in R} \Phi_k[(0123, r)(q, e)] = \Phi_k[(0123, r)(q, *)]$$

то есть среди двух кандидатов на второй ход p и q , достаточно рассмотреть лишь номер p , если будет найдено преобразование ψ или ϕ .

Реализуем, с помощью компьютерной программы, следующий алгоритм.

Инициализируем пустое множество S . Затем делаем цикл по всем ходам $p \in \Omega$ и перебираем все ходы $q \in S$. Если не существует преобразования ψ такого, что $\psi(0123) = 0123, \psi(p) = q$ или $\psi(0123) = q, \psi(p) = 0123$, тогда добавим номер p в множество S . В конце отбросим из множества S номер 0123, так как этот ход уже был сделан. В результате мы получим множество $S_2 = S \setminus (0123)$.

Утверждение доказано.

Примечание. Тоже самое верно и для функции Λ .

Теперь вместо перебора в качестве второго хода 5040 вариантов, достаточно рассмотреть лишь $|S_2| = 19$ вариантов.

1.6 Множество третьих ходов.

Проведя аналогичные рассуждения для третьего хода, получаем, что справедливо следующее утверждение:

Лемма 9 *Рассмотрим любой номер $s \in S_2$, и два номера p и q , тогда если существует ψ , такое что*

$$\psi(0123) = 0123, \psi(s) = s, \psi(p) = q,$$

то

$$\Phi_k[(0123, r_1)(s, r_2)(p, r_3)] = \Phi_k[(0123, r_1)(s, r_2)(q, r_3)]$$

По аналогии с построением множества вторых ходов, для каждого номера s из множества S_2 можно построить множество третьих ходов $S_3(s)$. В этом случае мы также отбросим из множества $S_3(s)$ сам номер s .

Все множества $S_3(s)$ содержат 7072 элемента, получается, в среднем по $7072/19=372.2$ номера на каждый элемент из S_2 . Теперь при переборе ходов с первого по третий перебирается всего 7072 номера, вместо $5039^2 = 2.5 \times 10^7$ вариантов.

Примечание. Тоже самое верно и для функции Λ .

1.7 Оценки подмножеств.

Лемма 10 $\forall R' \subset R$ и любой последовательности ходов $(t_1, r_1) \dots (t_n, r_n)$, и номера t справедливо утверждение

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, *)] \geq \sum_{r \in R'} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)]$$

в частности

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, *)] \geq \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)] \quad \forall r \in R$$

Доказательство. Используя формулу 1, получаем что

$$\begin{aligned} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, *)] &= \sum_{r \in R} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)] = \\ &= \sum_{r \in R'} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)] + \sum_{r \notin R'} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)] \geq \\ &\geq \sum_{r \in R'} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)] \end{aligned}$$

Утверждение доказано.

Примечание. Тоже самое верно и для функции Λ .

1.8 Минимально возможная оценка.

Минимизация числа семиходовок. Если в процессе работы алгоритма удалось найти ход, по которому оценка равна нулю, то сразу возвращаем лучшую оценку 0, так как улучшать дальше некуда.

Минимизация средней длины игры. Пусть множество оставшихся номеров состоит из $|S|$ элементов и делается k -й ход. Если некоторый номер делит множество оставшихся номеров на подмножества размером 1, тогда это лучший ход и сумма ходов равна $\Lambda = k + (|S| - 1)(k + 1) = |S|(k + 1) - 1$.

1.9 Быстрая оценка.

Пусть сделана последовательность ходов $(t_1, r_1) \dots (t_n, r_n)$. Обозначим за $S = (s_1, s_2 \dots)$ множество номеров, удовлетворяющих всем ходам из последовательности. Когда во множестве S остается мало элементов, в некоторых случаях, можно сразу получить оценку.

Семиходовки. Если делается седьмой ход.

$$\Phi_7[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} 1 & \text{если } |S| = 1 \\ 5040 & \text{если } |S| > 1 \end{cases}$$

Если делается шестой ход.

$$\Phi_7[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} & \text{если } |S| = 1 \text{ или } |S| = 2 \text{ или} \\ |S| - 1 & |S| = 3 \text{ и на один из ходов } s_i \\ & \text{два других дают разные ответы.} \end{cases}$$

Если делается ход с первого по пятый.

$$\Phi_7[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} & \text{если } |S| = 1 \text{ или } |S| = 2 \text{ или} \\ 0 & |S| = 3 \text{ и на один из ходов } s_i \\ & \text{два других дают разные ответы.} \end{cases}$$

В дальнейшем мы будем считать не только семиходовки но и шестиходовки, пятиходовки и так далее. Вышеприведенные формулы можно расширить.

Если делается k -й ход.

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} 1 & \text{если } |S| = 1 \\ 5040 & \text{если } |S| > 1 \end{cases}$$

Если делается $(k - 1)$ -й ход.

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} & \text{если } |S| = 1 \text{ или } |S| = 2 \text{ или} \\ |S| - 1 & |S| = 3 \text{ и на один из ходов } s_i \\ & \text{два других дают разные ответы.} \end{cases}$$

Если делается ход с первого по $(k - 2)$ -й.

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} & \text{если } |S| = 1 \text{ или } |S| = 2 \text{ или} \\ 0 & |S| = 3 \text{ и на один из ходов } s_i \\ & \text{два других дают разные ответы.} \end{cases}$$

Средняя длина игры. Если делается k -й ход и $|S| = 1$ или $|S| = 2$ или $|S| = 3$ и на один из ходов s_i два других дают разные ответы, тогда

$$\Lambda[(t_1, r_1) \dots (t_n, r_n)] = k + (|S| - 1)(k + 1) = |S|(k + 1) - 1$$

1.10 Отсечения для ходов 4-6.

Для ходов 1-3 выше были построены множества, которые помогают значительно ускорить алгоритм. По аналогии можно построить множества для ходов 4-6, но, к сожалению, они будут занимать слишком много оперативной памяти. Сейчас будет описано как отбросить некоторые номера, при рассмотрении ходов с четвертого по шестой.

Примечание. На практике все отсечения использовалось только для решения последовательности ходов (0123,0.1)(1245,0.0) точным алгоритмом.

1.10.1 Отсутствующие цифры.

Пусть сделана последовательность ходов $(t_1, r_1) \dots (t_n, r_n)$. Обозначим за $S = (s_1, s_2 \dots)$ множество номеров, удовлетворяющих всем ходам из последовательности. Пусть все номера $s_1, s_2 \dots$ не содержат некоторых цифр $D = (d_1, d_2 \dots)$. Будем считать, что отсутствующие цифры упорядочены по возрастанию $d_1 < d_2 < \dots$. Рассмотрим номер p , который включает в себя одну или несколько отсутствующих цифр. Тогда, если номер p , содержит только отсутствующие цифры, то его можно исключить из перебора - так как ничего нового он не даст. Если же номер p содержит от одной до трех отсутствующих цифр, тогда они должны идти в строгом порядке $d_1, d_2 \dots$, иначе этот номер можно отбросить. Поясним сказанное, на примере.

Пусть была сделана последовательность ходов (0123,0.1)(1245,0.0), тогда все номера из множества S не будут содержать цифр $D = (1, 2, 4, 5)$. Рассмотрим номер 4058. Он содержит две отсутствующие цифры 4 и 5. Этот номер отбросить, так как номер 1028, эквивалентен номеру 4058.

1.10.2 Незазванные цифры.

По аналогии с отсутствующими цифрами можно сказать тоже самое о незазванных цифрах. Пусть сделана последовательность ходов $(t_1, r_1) \dots (t_n, r_n)$ и есть множество незазванных цифр $A = (a_1, a_2 \dots)$. Будем считать, что незазванные цифры упорядочены по возрастанию, $a_1 < a_2 < \dots$. Также, как и для отсутствующих цифр, можно отбросить номера, где незазванные цифры не упорядочены по возрастанию. Рассмотрим пример.

Пусть сделаны ходы (0123,0.1)(1245,0.2), тогда множество незазванных цифр $A = (6, 7, 8, 9)$. Рассмотрим номер 7601, очевидно его можно отбросить, так как номер 6701 ему эквивалентен. Аналогично можно отбросить номер 7123, так как ему эквивалентен номер 6123.

Примечание. Единственное отличие от отсутствующих цифр состоит в том, что нельзя отбросить все номера, содержащие только незазванные

цифры. Среди всех номеров, содержащих только незазванные цифры, мы должны рассмотреть только один номер 6789.

1.10.3 Ходы разбивающие множество оставшихся номеров на одну группу.

Пусть сделана последовательность ходов $(t_1, r_1) \dots (t_n, r_n)$. Тогда, очевидно, что если некий номер разобьет множество оставшихся номеров на одну группу, то этот ход будет не лучшим и может быть отброшен.

1.11 На несколько ходов получены одинаковые ответы.

Рассмотрим последовательность ходов $(t_1, r_1) \dots (t_n, r_n)$. Разобьем все номера $t_1 \dots t_n$ на группы с одинаковыми ответами. Множество номеров с ответом b, c , где b число быков и c число коров обозначим за $T_{b,c}$.

Пример. Пусть задана последовательность, состоящая из трех ходов, $(0123, 0.1)(1234, 0.1)(5678, 0.2)$. Тогда $T_{0.1} = \{0123, 1234\}$, $T_{0.2} = \{5678\}$. Последовательность, заданную выше, можно представить в альтернативной форме $(T_{0.1}, 0.1)(T_{0.2}, 0.2)$.

Лемма 11 Пусть есть последовательность ходов $(t_1, r_1) \dots (t_n, r_n)$, имеющая представление в альтернативной форме $(T_{b_1, c_1}, b_1, c_1) \dots (T_{b_l, c_l}, b_l, c_l)$, и два номера p и q . Пусть существует преобразование ψ такое, что все множества номеров $T_{b,c}$ с одинаковыми ответами, переходят сами в себя, $\forall b, \forall c \psi(T_{b,c}) = T_{b,c}$ и номер p переходит в q $\psi(p) = q$. Тогда, для любого ответа r и любого k

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, r)] = \Phi_k[(t_1, r_1) \dots (t_n, r_n)(q, r)]$$

Доказательство. Используя лемму 1 и то, что порядок ходов неважен, получаем что

$$\begin{aligned} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, r)] &= \Phi_k[(T_{b_1, c_1}, b_1, c_1) \dots (T_{b_l, c_l}, b_l, c_l)(p, r)] = \\ &= \Phi_k[(\psi(T_{b_1, c_1}), b_1, c_1) \dots (\psi(T_{b_l, c_l}), b_l, c_l)(\psi(p), r)] = \\ &= \Phi_k[(T_{b_1, c_1}, b_1, c_1) \dots (T_{b_l, c_l}, b_l, c_l)(q, r)] = \Phi_k[(t_1, r_1) \dots (t_n, r_n)(q, r)] \end{aligned}$$

Утверждение доказано.

Примечание. То же самое верно и для функции Λ .

Лемма 11 является обобщением леммы 1, так как номера t_i уже не обязаны переходить сами в себя при преобразовании ψ . Достаточно лишь того, что все множества номеров с одинаковыми ответами переходят сами в себя. Используя лемму 11, можно еще уменьшить множества третьих ходов, когда на первые два хода получены одинаковые ответы.

2 Алгоритмы.

2.1 Точный алгоритм.

Точный алгоритм всегда будет давать точную оценку, он имеет на входе множество номеров S , удовлетворяющих всем предыдущим ходам, и верхнюю оценку β , о которой написано в секции об альфа-бета отсечениях. Алгоритм делает перебор по всем возможным ходам $t \in \Omega$, после каждого из них множество S разобьется на подмножества с одинаковыми ответами $S_1(t) \dots S_k(t)$, теперь надо найти оценку по всем этим подмножествам, рекурсивно вызвав этот же алгоритм. Оценка хода t будет суммой оценок всех подмножеств $S_1(t) \dots S_k(t)$. Минимум оценки по всем ходам и будет точной оценкой множества S .

2.1.1 Алгоритм последнего слоя.

Зная, что шестой ход - это последний ход, можно сильно ускорить алгоритм, поскольку мы должны сделать такой ход t , чтобы после него все подмножества $S_i(t)$ содержали максимум по одному элементу. Поэтому как только становится известно, что для какого-то i $|S_i(t)| > 1$ - ход t заведомо не лучший. Если число оставшихся номеров больше 14, тогда невозможно решить все номера за семь или менее ходов и можно сразу вернуть максимально возможную оценку. Если осталось ровно 14 номеров, тогда достаточно перебирать лишь номера из множества S .

Этот алгоритм всегда дает точную оценку, и используется всегда.

Семиходовки. Сначала в качестве шестого хода переберем номера из множества S . Если ход разбивает множество на группы по одному элементу $|S_1(t)| = \dots = |S_n(t)| = 1$, то сразу же можно сказать, что это лучший ход и число семиходовок равно $|S| - 1$.

Затем переберем остальные ходы. Если ход разбивает S на группы по одному элементу, то он лучший и число семиходовок равно $|S|$.

Если же так и не оказалось хода, дробящего множество S на группы по одному элементу, значит возвращается оценка 5040, которая заведомо хуже любой другой.

Средняя длина игры. Сначала в качестве шестого хода переберем номера из S . Если ход разбивает множество S на группы по одному элементу, то он лучший и оценка будет равна $\Lambda = 6 + 7(|S| - 1) = 7|S| - 1$.

Затем переберем остальные ходы. Если ход разбивает множество S на группы по одному элементу, то он лучший и оценка равна $\Lambda = 7|S|$.

Если же так и не оказалось хода, дробящего множество S на группы по одному элементу, значит возвращается оценка 5040×7 , которая заведомо хуже любой другой.

2.2 Эвристические алгоритмы.

Эвристические алгоритмы не всегда возвращают точную оценку, но они намного быстрее точного алгоритма. Поэтому мы будем последовательно сужать область их использования, постепенно переходя к точному алгоритму. Для поиска минимального числа семиходов дробящий алгоритм показал лучшие результаты, для поиска минимальной длины игры эффективнее алгоритм Лармауса.

2.2.1 Дробящий алгоритм.

Дробящий алгоритм используется для поиска минимального числа семиходов.

Пусть у нас есть некоторое множество номеров $S = (s_1, s_2 \dots)$, удовлетворяющих всем предыдущим ходам.

Будем перебирать всевозможные ходы t . Каждый из них разобьет множество S на подмножества с одинаковыми ответами $S_1(t) \dots S_k(t)$. Обозначим за $n_i(t) = |S_i(t)|$ ($\sum n_i(t) = |S|$). В дальнейшем будем считать, что подмножества $S_i(t)$ упорядочены по размеру, то есть $n_1(t) \geq n_2(t) \geq \dots$

Пусть для хода t получен набор $n_1(t), n_2(t) \dots$, а для хода p получен набор $n_1(p), n_2(p) \dots$. Будем считать, что ход t лучше, чем p если $n_1(t) < n_1(p)$ или $n_1(t) = n_1(p)$ и $n_2(t) < n_2(p)$ и так далее. Например, ход с набором $18, 18, 17 \dots$ лучше, чем ход с набором $18, 18, 18 \dots$. Дробящий алгоритм выбирает ход, который лучше всего дробит множество номеров S .

Дополнительно будем использовать следующие правила.

- Если найден ход t , такой что $n_1(t) = 1$, то
 1. если $t \in S$ это самый лучший ход.
 2. если $t \notin S$, то дальнейший поиск ведется только по ходам из S .
- Если имеется два хода $t \in S$ с набором $n_1(t) \dots n_k(t)$ и $p \notin S$ с набором $n_1(p) \dots n_k(p)$ такие что $n_1(t) = n_1(p) \dots n_k(t) = n_k(p)$, тогда лучшим ходом будем считать ход t . Это означает, что среди ходов с одинаковыми наборами предпочитаем ходы из множества S .

2.2.2 Алгоритм Лармауса.

Алгоритм Лармауса используется для минимизации средней длины игры.

Пусть был сделан ход t и множество номеров S разбилось на подмножества $S_1(t) \dots S_k(t)$, с размерами $n_i(t) = |S_i(t)|$. Введем функцию принадлежности

$$IN(t, S) = \begin{cases} 1 & \text{если } t \in S \\ 0 & \text{если } t \notin S \end{cases}$$

Тогда лучшим ходом, будет тот, который минимизирует функцию

$$F(t) = \sum_{n_i(t) > 1} n_i(t) \log(n_i(t)) - 2 \log 2 \times IN(t, S)$$

2.3 Ускорения алгоритмов.

По аналогии с алгоритмом последнего слоя, можно сразу вернуть максимальную оценку на предпоследнем слое, если оставшихся номеров больше чем $1 + 13 + 13^2$.

Перед тем, как делать поиск по всем возможным номерам, сделаем поиск только среди номеров, оставшихся в множестве, рекурсивно вызвав алгоритм поиска. Если полученная таким образом оценка e будет меньше чем верхняя оценка β , то верхнюю оценку можно снизить $\beta = e$.

2.4 Обозначения алгоритмов.

В дальнейшем используем следующие обозначения алгоритмов.

Минимизация числа семиходовок.

1. crush35 - дробящий алгоритм используется для ходов 3-5, для остальных ходов используется точный алгоритм.
2. crush45 - дробящий алгоритм используется для ходов 4, 5.
3. crush5 - дробящий алгоритм используется для хода 5.
4. exact - используется только точный алгоритм.

Минимизация средней длины игры.

1. avg35 - алгоритм Лармауса используется для ходов 3-5, для остальных ходов используется точный алгоритм.
2. avg45 - алгоритм Лармауса используется для ходов 4, 5.
3. avg5 - алгоритм Лармауса используется для хода 5.

Примечание. Во всех алгоритмах для шестого хода используется точный алгоритм.

2.5 Расширение для подсчета шестиходовок, пятиходовок и так далее.

Примечание. Этот раздел касается только алгоритмов минимизации семиходовок.

В дальнейшем для различных алгоритмов будут построены деревья, для онлайн игры с человеком. Довольно часто будет возникать ситуация, что число семиходовок равно нулю, в этом случае можно найти ход, который

минимизирует число шестиходовок, если же и число шестиходовок можно сделать равным нулю, то минимизируются пятиходовки и так далее.

Алгоритм оценки написан таким образом, что он может минимизировать не только семиходовки, но и шестиходовки, пятиходовки и так далее. На входе он, кроме множества оставшихся ходов и верхней оценки β , имеет еще два дополнительных параметра:

count - показывающий, что считать: семиходовки, шестиходовки. . .

mde - показывающий, максимальную глубину, до которой необходимо использовать точный алгоритм, при этом последний слой всегда использует точный алгоритм. Если мы считаем семиходовки, то для хода 6 используется алгоритм последнего слоя, если шестиходовки, то он используется для хода 5 и так далее.

Примечание. Если алгоритм не может угадать все номера за нужное нам число ходов, то он возвращает максимальную оценку.

3 Компоненты алгоритма.

3.1 Альфа-бета отсечения.

Альфа-бета отсечения широко используются в задачах перебора для различных игр. Этот алгоритм очень сильно ускоряет поиск, при этом, получаемая оценка остается точной. Более подробную информацию можно найти в интернете.

3.2 Массивы второго, третьего ходов и фиксированный первый ход.

Используя теоретические расчеты, нам достаточно использовать лишь ход 0123 в качестве первого хода, номера из множества $s \in S_2$ в качестве второго хода, а также номера из множеств $S_3(s)$, $s \in S_2$ в качестве третьего хода.

3.3 Упорядочивание ходов 2 и 3.

Как известно, перебор „хороших“ ходов раньше „плохих“ ускоряет алгоритм, поэтому, сохраняя множества вторых и третьих ходов S_2 и $S_3(s_2)$, предварительно переупорядочим их в обратном направлении.

3.4 Упорядочивание остальных ходов.

Ходы 4-6 также будем перебирать в обратном порядке.

3.5 Таблица ответов.

Поскольку в алгоритме нужно будет часто обращаться к функции ответа номера a на номер b , то для вычисления числа быков и коров, мы предварительно выделим память под массив типа `char` размера 5040×5040 и до

запуска расчетов заполним его ответами. Эта таблица занимает примерно 24.2 мегабайта.

3.6 Механизм задач.

Для проблем, которые занимают длительное время, был разработан механизм задач. Вначале создается файл jobs.txt, в котором в каждой строке описаны параметры задачи. Ниже приведен пример файла jobs.txt.

```
solve 0123(0,1)+0145(0,1) 7 4 8 #
solve 0123(0,1)+0134(0,1) 7 4 28 #
solve 0123(0,2)+1234(0,1)+4532(0,1) 7 5 3 #
```

Вначале идет ключевое слово solve, которое обозначает, что эта задача еще не взята для решения. В процессе решения заданий различные процессы берут задачи из файла и заменяют слово solve на taken, чтобы показать, что эта задача уже решается. Затем идет последовательность ходов с ответами, которую нужно решить. После этого идут три параметра алгоритма.

- *count* - показывает, что необходимо считать: 7 - семиходовки, 6 - шестиходовки и так далее.

Примечание. Этот параметр не используется при минимизации средней длины игры.

- *mde* - определяет с какого хода начать использовать дробящий алгоритм: 2 - с третьего, 3 - с четвертого . . .
- β - верхняя оценка бета.

Механизм задач, позволяет параллельно решать различные задания, несколькими нитями. Он реализован таким образом, что возобновляет решение задач после выключения компьютера, таким образом задачи могут считаться несколько дней.

Если необходимо выполнить несколько задач, результаты которых зависят друг от друга, то будет лучше если, первыми будут идти задачи, которые по интуитивным предположениям, будут считаться дольше. Это позволяет максимально загрузить процессор. Когда возникает ситуация, что невыполненных заданий уже нет, то все нити, кроме одной, закончили свою работу и ждут. Чем за более короткое время будет выполняться последняя задача тем лучше. Поэтому лучше предварительно, хотя бы на интуитивном уровне, отсортировать задачи по убыванию времени выполнения.

4 Компоненты программы.

Программа состоит из нескольких приложений. Все они написаны на с++, кроме одного, которое написано на javascript и используется для онлайн игры, где компьютер отгадывает номер, задуманный человеком.

4.1 Консольное приложение executor.

Приложение используется для расчета задач, записанных в файле jobs.txt.

4.2 Сервис windows.

Запуск нескольких нитей executor'a.

Расчеты для алгоритмов crush5, avg5, exact занимают очень долгое время. В связи с этим был написан системный сервис windows, который автоматически запускался при загрузке компьютера. Поскольку компьютер где проводились расчеты имел четыре процессора, то запускалось четыре нити с низким приоритетом, чтобы не мешать остальным приложениям.

4.3 Приложение под windows - bcw.

Вспомогательные функции.

- Создание заданий и сбор данных для алгоритма перехода (см. 5.2).
- Решение с помощью алгоритмов crush35, crush45, avg35, avg45.
- Построение деревьев.
- Загрузка дерева из файла, получение статистики по нему, создание строки инициализации дерева для javascript.
- Построение html файла для обзора дерева через jQuery.
- Решение последовательности $(0123, 0.1)(1245, 0.0)$ точным алгоритмом для минимизации семиходовок.

4.4 Консольное приложение shutdown.

Выключение компьютера в заданное время.

В связи с тем, что некоторые задачи считаются очень долго, а ночью электричество примерно в три раза дешевле, чем днем, компьютер множество раз рассчитывал задачи ночью. Это приложение автоматически выключает компьютер когда, счетчик электричества переходит на дневной тариф.

4.5 Приложение tree на javascript.

Игра с человеком через web интерфейс.

- Загрузка и получение статистики для дерева.
- Игра с человеком, используя сохраненное дерево.

5 Результаты минимизации семиходовок.

5.1 Алгоритмы crush35 и crush45.

Так как дробящий алгоритм, во много раз быстрее точного запустим сначала алгоритм crush35. Затем, только для случаев, где число семиходовок больше нуля запустим алгоритм crush45 с верхними оценками, полученными от алгоритма crush35. В будущем аналогично поступим для алгоритмов crush5 и exact. Число семиходовок приведено в таблице.

число номеров	первый ход	crush35	crush45
1440	0123 0.1	76	53
1260	0123 0.2	22	10
720	0123 1.1	1	0
480	0123 1.0	0	-
360	0123 0.0	0	-
264	0123 0.3	0	-
216	0123 1.2	0	-
180	0123 2.0	0	-
72	0123 2.1	0	-
24	0123 3.0	0	-
9	0123 0.4	0	-
8	0123 1.3	0	-
6	0123 2.2	0	-
1	0123 4.0	0	-
всего 5040		99	63

Алгоритм crush35 ни в одном из вариантов ответа на первый ход 0123 не вернул оценку 5040, а это означает, что он отгадывает все номера за семь или менее ходов.

5.2 Переход от одного алгоритма к другому.

Получение оценок для алгоритмов crush5 и exact сделаем в два этапа. Сначала, используя результаты алгоритма crush45, получим оценки для crush5, затем, используя результаты алгоритма crush5, сделаем переход к алгоритму exact. Расчет алгоритмом crush5, а уж тем более алгоритмом exact будет занимать много времени, поэтому решение этими алгоритмами разобьем на несколько шагов.

Назовем алгоритм от которого мы переходим prev, а тот, к которому мы переходим - next.

Шаги 1-5. Оценка лучших ходов, полученных алгоритмом prev.

1. Для алгоритма prev находим лучший ход t_1 для первого хода (0123,0.1).

2. Рассмотрим все ответы r_1 такие что, оценка числа семиходовок алгоритмом prev больше нуля $\Phi_7^{\text{prev}}[(0123, 0.1)(t_1, r_1)] > 0$.
3. Для таких ответов r_1 считаем число семиходовок алгоритмом next $e_1(r_1) = \Phi_7^{\text{next}}[(0123, 0.1)(t_1, r_1)]$, при этом в качестве верхней оценки β используем оценку от алгоритма prev $\beta = \Phi_7^{\text{prev}}[(0123, 0.1)(t_1, r_1)]$.
4. Обозначим $E_1 = \sum_{r_1} e_1(r_1)$
5. Проведя шаги 1-4, для первого хода $(0123, 0.2)$, получим лучший ход t_2 и его оценку E_2 . Таким образом, шаги 1-5 берут лучшие ходы для алгоритма prev и находят для них оценку алгоритмом next .

Шаг 6. Построение таблицы для ходов $(0123, 0.1)(t, 0.2)$.

6. Запускаем алгоритм next по всем вторым ходам t кроме $(t_1, t_2, 0132, 0231, 1032, 1230)$, для последовательности $(0123, 0.1)(t, 0.2)$. По ходам t_1, t_2 оценки известны. Также мы заведомо предполагаем, что номера $0132, 0231, 1032, 1230$ уже не будут лучшими ходами, по ним будем запускать алгоритм next в самом конце. При запуске алгоритма next используем верхнюю оценку $\beta = \min(E_1, \Phi_7^{\text{prev}}[(0123, 0.1)(t, 0.2)])$. Полученную оценку, обозначим за $e_{12}(t) = \Phi_7^{\text{next}}[(0123, 0.1)(t, 0.2)]$. Отсортируем ходы по возрастанию $e_{12}(t)$ и выпишем их в таблицу в которую добавим номера t_1 и t_2 с их оценками.

Примечание. Сортируя номера в шаге 6 мы фактически сортируем номера (или задачи) в порядке убывания времени выполнения, тем самым дополнительно ускоряя программу (см. раздел 3.6).

Примечание. Построив таблицу для ходов $(0123, 0.1)(t, 0.2)$, мы можем ее использовать для нахождения оптимально хода, как для первого хода $(0123, 0.1)$, так и для первого хода $(0123, 0.2)$ (см. лемму 3).

Шаги 7-8. Поиск оценки числа семиходовок для первого хода $(0123, 0.2)$.

7. Рассмотрим ход t с минимальной оценкой $e_{12}(t)$. Для него найдем другие ответы r , по которым число семиходовок при использовании алгоритма next , для последовательности $(0123, 0.2)(t, r)$ больше нуля. Для ускорения процесса используем оценки полученные алгоритмом prev . Обозначим за E_2^* сумму всех таких оценок. E_2^* - оценка последовательности $(0123, 0.2)(t, *)$ алгоритмом next . Если $E_2^* < E_2$, то уменьшим лучшую оценку $E_2 = E_2^*$.
8. Для номеров t у которых $e_{12}(t) < E_2$, делаем аналогичную процедуру, как на предыдущем шаге и потенциально снижаем E_2 . Результирующая оценка E_2 и будет лучшей оценкой для алгоритма next , после хода $(0123, 0.2)$.

Шаг 9. Поиск оценки числа семиходовок для первого хода $(0123, 0.1)$.

9. Тоже самое надо сделать для первого хода $(0123, 0.1)$.

Шаг 10. Проверка оставшихся номеров.

10. Убедимся, что ходы $t = (0132, 0231, 1032, 1230)$ не лучшие, для этого запустим алгоритм next для $(0123, 0.1)(t, 0.1)$ с верхней оценкой $\beta = E_1$ и $(0123, 0.2)(t, 0.2)$ с $\beta = E_2$

Теперь сделаем практический переход от алгоритма crush45 к алгоритму crush5.

5.3 Алгоритм crush5.

5.3.1 Расчет для лучших ходов (шаги 1-5).

Рассмотрим лучший номер для первого хода $(0123, 0.1)$, полученный алгоритмом crush45 - это 1245. После него может быть несколько ответов, но только по трем из них число семиходов больше нуля, по этим случаям запустим алгоритм crush5.

ход 1	ход 2	crush45	crush5
0123 0.1	1245 0.1	41	38
0123 0.1	1245 0.2	10	7
0123 0.1	1245 0.0	2	1
всего		53	46

Аналогично, если первый ход $(0123, 0.2)$, то лучший номер - 1435.

ход 1	ход 2	crush45	crush5
0123 0.2	1435 0.1	9	8
0123 0.2	1435 0.2	1	0
всего		10	8

Теперь у нас есть стартовые оценки на максимальное число семиходов. После первого хода $(0123, 0.1)$ - $E_1 = 46$, после хода $(0123, 0.2)$ - $E_2 = 8$.

5.3.2 Построение таблицы (шаг 6).

t	(0123,0.1)(t,0.2)
1234	3
1034	4
1245	7
1204	7
1435	8
1045	9
1456	10
0234	10
0245	15
0214	17
0134	18
0456	30
0145	39
0124	41
4567	≥ 46

5.3.3 Лучший ход после (0123,0.2) (шаги 7-8).

t	(0123,0.1)(t,0.2)	(0123,0.2)(t,0.2)	$E_2 = 8$
1234	3	1	$E_2 = 4$
1034	4	×	
1245	7	×	
1204	7	×	
1435	8	0	
1045	9	×	
1456	10	×	
0234	10	×	
0245	15	×	
0214	17	×	
0134	18	×	
0456	30	×	
0145	39	×	
0124	41	×	
4567	≥ 46	×	

Рассмотрим второй ход 1234, только для вторых ответов 0.1 и 0.2 число

семиходовок больше нуля.

ход 1	ход 2	crush45	crush5
0123 0.2	1234 0.1	-	3
0123 0.2	1234 0.2	-	1
0123 0.2	1234 1.1	1	0
всего			4

Таким образом если второй ход 1234, то число семиходовок равно 4. Из таблицы видно, что рассматривать остальные номера не имеет смысла, так как по ним число семиходовок будет не меньше, чем 4. Таким образом лучший второй ход, для алгоритма crush5 - 1234, число семиходовок - 4.

5.3.4 Лучший ход после (0123,0.1) (шаг 9).

t	(0123,0.1)(t,0.2)	(0123,0.1)(t,0.1)	$E_1 = 46$
1234	3	≥ 43	$E_1 \geq 46$
1034	4	≥ 42	$E_1 \geq 46$
1245	7	38	$E_1 = 46$
1204	7	≥ 39	$E_1 \geq 46$
1435	8	≥ 38	$E_1 \geq 46$
1045	9	≥ 37	$E_1 \geq 46$
1456	10	≥ 36	$E_1 \geq 46$
0234	10	≥ 36	$E_1 \geq 46$
0245	15	≥ 31	$E_1 \geq 46$
0214	17	≥ 29	$E_1 \geq 46$
0134	18	≥ 28	$E_1 \geq 46$
0456	30	≥ 16	$E_1 \geq 46$
0145	39	≥ 7	$E_1 \geq 46$
0124	41	≥ 5	$E_1 \geq 46$
4567	≥ 46	\times	

После второго хода 1245, полученного как лучший ход алгоритмом crush45, ни один другой номер не дал лучший результат.

5.3.5 Проверка номеров с цифрами 0-3 (шаг 10).

В самом конце необходимо убедиться, что номера которые содержат только цифры от 0 до 3, не будут лучшими ходами.

ход 1	ход 2	crush5
0123 0.1	0132 0.1	≥ 46
0123 0.1	0231 0.1	≥ 46
0123 0.1	1032 0.1	≥ 46
0123 0.1	1230 0.1	≥ 46

ход 1	ход 2	crush5
0123 0.2	0132 0.2	≥ 4
0123 0.2	0231 0.2	≥ 4
0123 0.2	1032 0.2	≥ 4
0123 0.2	1230 0.2	≥ 4

5.3.6 Результаты алгоритма crush5.

После хода (0123,0.1) лучший второй ход - 1245, число семиходовок - 46.
После хода (0123,0.2) лучший второй ход - 1234, число семиходовок - 4.

5.4 Точный алгоритм.

К сожалению для расчета точным алгоритмом требуется длительное время. В данное время ведется поиск серверов для завершения расчетов.

5.5 Сравнительная таблица результатов алгоритмов.

ход 1	crush35	crush45	crush5
0123 0.1	76	53	46
0123 0.2	22	10	4
0123 1.1	1	0	0
всего 7-ходовок	99	63	50

6 Минимизация числа шестиходовок, пятиходовок и так далее.

Выше мы рассматривали, только ответы на первый ход 0123 при которых число семиходовок больше нуля. Это только два ответа 0.1 и 0.2. Для остальных ответов мы можем минимизировать число шестиходовок. Если число шестиходовок будет равно нулю, можно минимизировать число пятиходовок и так далее. Задача по поиску минимального числа шестиходовок намного легче основной задачи, так как теперь рассматривается на один ход меньше. Таблица минимизации, представлена ниже, в ней отброшен ответ 4.0.

ход 1	7-ходовки	6-ходовки	5-ходовки	4-ходовки
0123 0.1	≤ 46			
0123 0.2	≤ 4			
0123 1.1	0	213		
0123 1.0	0	88		
0123 0.0	0	84		
0123 0.3	0	20		
0123 1.2	0	8		
0123 2.0	0	4		
0123 2.1	0	0	28	
0123 3.0	0	0	2	
0123 0.4	0	0	0	6
0123 1.3	0	0	1	
0123 2.2	0	0	0	4

7 Результаты минимизации длины игры.

Задача минимизации средней длины игры сложнее задачи минимизации семиходовок, но точная оценка для нее известна (см. [1] and [2]). Таким образом достаточно найти алгоритм, дающий оптимальную оценку. Вначале запустим алгоритмы avg35, avg45. После этого запустим очень быстрый алгоритм, который не использует эвристические алгоритмы, но в качестве ходов 4-6 перебирает только оставшиеся номера (столбец fso).

Возьмем результаты точного алгоритма из [2] и запишем их в столбец exact. Если посмотреть таблицу ниже, и сравнить минимальную оценку алгоритмов avg45, fso и точного алгоритма (столбец exact), мы увидим, что сейчас оптимальный результат не достигнут только по ответам 0.1, 0.2 и 1.1. Поэтому, только для них, запустим алгоритм avg5.

ход 1	avg35	avg45	fso	min(avg45,fso)	exact	avg5
0123 0.0	1808	1807	1806	1806	1806	1806
0123 0.1	8009	7951	7942	7942	7935	7935
0123 0.2	6828	6817	6818	6817	6808	6808
0123 0.3	1284	1268	1269	1268	1268	1268
0123 0.4	32	32	32	32	32	32
0123 1.0	2400	2394	2393	2393	2393	2393
0123 1.1	3731	3717	3716	3716	3712	3712
0123 1.2	1031	1020	1020	1020	1020	1020
0123 1.3	30	30	30	30	30	30
0123 2.0	845	839	839	839	839	839
0123 2.1	314	312	312	312	312	312
0123 2.2	21	21	21	21	21	21
0123 3.0	97	97	97	97	97	97
0123 4.0	1	1	1	1	1	1
всего ходов	26 431	26 306	26 296	26 294	26 274	26 274
среднее	5.2442	5.2194	5.2175	5.2171	5.2131	5.2131

Из результирующей таблицы видно, что алгоритм avg5 уже дает точную оценку. Таким образом минимально возможная длина игры достигнута.

8 Построение дерева ходов.

Для того чтобы создать программу, угадывающую загаданные человеком номера и не использующую длительные расчеты, необходимо предварительно построить дерево ходов и сохранить его в файл. Затем в отдельной программе можно загрузить это дерево и использовать его без всяких расчетов.

Минимизация семиходовок. При построении дерева мы будем не только минимизировать семиходовки. Если число семиходовок для узла дерева

равно нулю, то мы минимизируем число шестиходовок, если же можно сделать ход, такой что число шестиходовой будет равно нулю - будем минимизировать пятиходовки и так далее. При этом, для минимизации шестиходовок, пятиходовок и так далее мы будем всегда использовать точный алгоритм. Для минимизации же семиходовок, для первых ходов (0123,0.1) и (0123,0.2), используем один из вышеприведенных алгоритмов `crush45`, `crush5`. Назовем такие алгоритмы оптимизированными и будем обозначать их `crush45o`, `crush5o`. Эти алгоритмы будут давать столько же семиходовок, что и их неоптимизированные варианты, но они будут существенно уменьшать число шестиходовок, пятиходовок и так далее.

При создании дерева будем использовать предварительно рассчитанные первые три хода. В процессе построения, для каждого узла дерева сохраним также дополнительный параметр $n - th$. Он будет обозначать с чего начинать считать потомкам. Например, если $n - th = 5$, то все потомки начинают считать с пятиходовок. Также будем сохранять саму оценку e , которая поможет при расчете в процессе построения дерева. Например, $n - th = 5$ и $e = 30$, обозначают, что суммарное число пятиходовок у всех потомков равно 30.

Минимизация средней длины игры. В процессе построения, для каждого узла дерева сохраним также оценку e , которая поможет при расчете в процессе построения дерева. Например, $e = 30$, означает, что суммарное число оценок у всех потомков равно 30. Также, как и при минимизации семиходовок, используются предварительно рассчитанные первые три хода.

8.1 Реализация алгоритма на с++.

Для проверки построенного дерева сначала использовалась программа на с++. Дерево загружалось, после чего прогонялось по всем возможным заданным номерам, до тех пор пока не был получен ответ 4.0. Затем собиралась статистика, сколько номеров угадано точно за 7, 6... ходов, а также сколько суммарно ходов потрачено на отгадывание всех номеров.

Также, с помощью программы, строилось дерево в виде html файла, которое можно просматривать используя браузер. Пример можно посмотреть здесь - <http://bulls-cows.sourceforge.net/crush5oTree.html>.

8.2 Реализация алгоритма на javascript.

После успешной проверки построенных деревьев на с++, можно создать web реализацию алгоритма. Чтобы не нагружать сервер создадим программу на языке javascript.

В javascript не используются указатели. Поэтому каждому элементу дерева присвоим уникальный номер id . Вместо указателей на дочерние элементы будем использовать их идентификаторы id .

Нашей целью является максимально уменьшить размер скрипта, поэтому будем использовать следующее

1. Дерево задается одной строкой, в качестве разделителя между элементами дерева используем символ „!“.
2. Будем сохранять все элементы последовательно, то есть первый элемент будет иметь $id = 0$, второй $id = 1$ и так далее, таким образом id не передается как параметр. Все, что сохраняется, это индекс лучшего хода (от 0 до 5039) и массив id потомков, по всем возможным ответам.
3. Каждый элемент дерева содержит элементы потомки, мы предварительно занумеруем все ответы всего их может быть 14, при этом не будем использовать ответ 4.0, то есть всего используем только 13 id потомков.
4. Для сокращения записи, передаем все параметры используя символы от ASCII кода 35 до 127, при этом отбрасываем символ „\“, который имеет ASCII код 92, так как для его передачи в строке javascript требуется 2 символа. Таким образом используется 92-ичная система счисления. Коды символов больше чем 127 не используются, так как они по разному распознаются различными браузерами. Максимальное передаваемое число меньше чем 92×92 , поэтому для передачи любого числа нам достаточно двух байт.
5. Если элемент не содержит потомков, то массив id потомков не передается.
6. Массив id потомков можно передавать двумя способами. Первый - указать все элементы, при этом всего будет передано 13 параметров по два байта каждый. Второй - передавать его парами „индекс массива (один байт от 0 до 13) и значение (2 байта)“. Число байт, при такой передаче всегда кратно трем, в то время как при передаче всех потомков используется 26 байт. Таким образом, всегда можно определить, каким способом был передан массив. При сохранении элемента дерева сначала посчитаем какой способ передачи будет содержать более короткую строку, и таким способом передадим массив.

В результате всех этих оптимизаций размер строки для одного дерева снизился с 222 килобайт до 31.6 килобайт.

9 Время расчетов.

9.1 Алгоритмы `crush35` и `crush45`.

Пример строки -

```
crush35 (0,1) max=5040 e=76 time 00:00:01 best=1456
```

означает, что запущен алгоритм `crush35` после хода (0123,0.1) с начальной оценкой 5040. В результате получена новая оценка 76 с лучшим ходом 1456, при этом время расчета заняло одну секунду.

```

solving 0123(*.*) count 7th
-----
crush35 (0.0) max=5040 e= 0 time 00:00:00 best=4567
crush35 (0.1) max=5040 e=76 time 00:00:01 best=1456
crush35 (0.2) max=5040 e=22 time 00:00:01 best=1234
crush35 (0.3) max=5040 e= 0 time 00:00:00 best=4567
crush35 (0.4) max=5040 e= 0 time 00:00:00 best=1456
crush35 (1.0) max=5040 e= 0 time 00:00:00 best=0134
crush35 (1.1) max=5040 e= 1 time 00:00:00 best=1456
crush35 (1.2) max=5040 e= 0 time 00:00:00 best=4567
crush35 (1.3) max=5040 e= 0 time 00:00:00 best=1456
crush35 (2.0) max=5040 e= 0 time 00:00:00 best=4567
crush35 (2.1) max=5040 e= 0 time 00:00:00 best=4567
crush35 (2.2) max=5040 e= 0 time 00:00:00 best=1456
crush35 (3.0) max=5040 e= 0 time 00:00:00 best=4567
total 99 time 00:00:03
-----
crush45 (0.1) max= 76 e=53 time 00:05:31 best=1245
crush45 (0.2) max= 22 e=10 time 00:02:31 best=1435
crush45 (1.1) max= 1 e= 0 time 00:00:04 best=1456
total 63 time 00:08:07

```

После прогона алгоритма `crush35`, для случаев, где число семиходовок больше нуля запускаем алгоритм `crush45`, с верхней оценкой, полученной алгоритмом `crush35`.

9.2 Алгоритм `crush5`.

Алгоритм `crush5` запускался с помощью механизма задач, поэтому по нему нет данных о времени выполнения.

9.3 Алгоритмы `avg35` и `avg45`.

Пример строки -

```
avg35 (0.0) max=**** e=1808 time 00:00:01 best=4567
```

означает, что запущен алгоритм `avg35` после хода (0123,0.0) с максимальной начальной оценкой. В результате получена новая оценка 1808 с лучшим ходом 4567, при этом время расчета заняло одну секунду.

```

solving 0123(*.*)
-----
avg35 (0.0) max=**** e=1808 time 00:00:01 best=4567
avg35 (0.1) max=**** e=8009 time 00:00:02 best=1245
avg35 (0.2) max=**** e=6828 time 00:00:05 best=1435
avg35 (0.3) max=**** e=1284 time 00:00:01 best=1245

```

```

avg35 (0.4) max=**** e= 32 time 00:00:00 best=1230
avg35 (1.0) max=**** e=2400 time 00:00:01 best=0456
avg35 (1.1) max=**** e=3731 time 00:00:03 best=0145
avg35 (1.2) max=**** e=1031 time 00:00:00 best=0245
avg35 (1.3) max=**** e= 30 time 00:00:00 best=1234
avg35 (2.0) max=**** e= 845 time 00:00:00 best=0456
avg35 (2.1) max=**** e= 314 time 00:00:00 best=0245
avg35 (2.2) max=**** e= 21 time 00:00:00 best=0132
avg35 (3.0) max=**** e= 97 time 00:00:00 best=0245
avg35 (4.0) max=**** e= 1 time 00:00:00 best=0123
total 26 431      avg=5.2442 time 00:00:17
-----
avg45 (0.0) max=1808 e=1807 time 00:08:23 best=4567
avg45 (0.1) max=8009 e=7951 time 00:29:51 best=1456
avg45 (0.2) max=6828 e=6817 time 00:32:22 best=1435
avg45 (0.3) max=1284 e=1268 time 00:06:30 best=1435
avg45 (0.4) max= 32 e= 32 time 00:00:00 best=-001
avg45 (1.0) max=2400 e=2394 time 00:11:44 best=0456
avg45 (1.1) max=3731 e=3717 time 00:19:45 best=0245
avg45 (1.2) max=1031 e=1020 time 00:05:22 best=0245
avg45 (1.3) max= 30 e= 30 time 00:00:01 best=-001
avg45 (2.0) max= 845 e= 839 time 00:03:48 best=0245
avg45 (2.1) max= 314 e= 312 time 00:01:16 best=0145
avg45 (2.2) max= 21 e= 21 time 00:00:00 best=-001
avg45 (3.0) max= 97 e= 97 time 00:00:15 best=-001
avg45 (4.0) max= 1 e= 1 time 00:00:00 best=0123
total 26 306      avg=5.2194 time 01:59:22

```

9.4 Алгоритм avg5.

Алгоритм avg5 запускался с помощью механизма задач, поэтому по нему нет данных о времени выполнения.

10 История редакций.

- Первая редакция 25 декабря 2008
- Вторая редакция 1 ноября 2011
 - Статья сильно расширена и дополнена.
 - Статья переведена на английский язык.
 - Создан сайт проекта на английском языке.

Список литературы

- [1] John Francis, Strategies for playing MOO, or Bulls and Cows.
<http://www.jfwaf.com/Bulls%20and%20Cows.pdf>
- [2] Tetsuro Tanaka, An Optimal MOO Strategy, Game Programming Workshop - Japan.