Посов Илья Александрович

РЕШЕНИЕ ЯПОНСКИХ КРОССВОРДОВ С ИСПОЛЬЗОВАНИЕМ КОНЕЧНЫХ АВТОМАТОВ

От редакции: статья продолжает цикл, посвященных системе DADemo демонстрации алгоритмов дискретного анализа, созданной под руководством профессора И.В. Романовского.

Японский кроссворд – это популярная головоломка, которую можно найти в большинстве современных газет и журналов. Японские кроссворды были изобретены в Японии где-то всего 20 лет назад. Иногда их называют иначе: нонограммы, гриддлеры или «рисуй по числам».

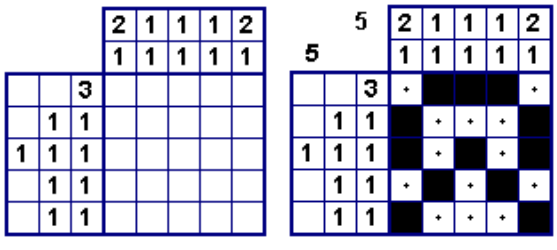
В этой статье мы посмотрим, что такое японские кроссворды, посмотрим, как их решают люди, и разберём алгоритм автоматического решения кроссвордов, основанный на конечных автоматах. Для демонстрации алгоритма мы будем пользоваться демонстрационной программой из системы DADemo.

Задание головоломки состоит в том, чтобы восстановить зашифрованный рисунок. На рис. 1*а* приведён пример японского кроссворда, на рис. 1*б* приведено его решение.

Восстановить рисунок означает разобраться, какие клетки таблицы чёрные, а какие белые. Для этого каждая строка и столбец имеют подсказки из нескольких чисел. Числа описывают, какие группы из чёрных клеток имеются в данной строке или столбце. Например, описание первого столбца – это числа (2, 1), они означают, что в этом столбце сначала идет группа из двух чёрных клеток, а потом из одной. Между двумя группами чёрных клеток обязательно должна быть хотя бы одна белая клетка.

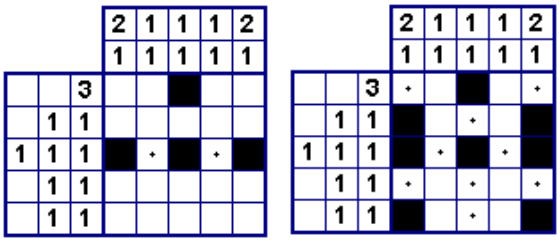
Посмотрим, как человек решит такую головоломку. Во-первых, можно заметить, что в третьей строке расставить блоки можно единственным способом. В первой строке при любом расположении блока из трех клеток, средняя клетка будет занята. Это дает частичное решение (рис. 2*а*).

Далее, можно заметить, что в третьем столбце с описанием (1, 1) уже обозначе-



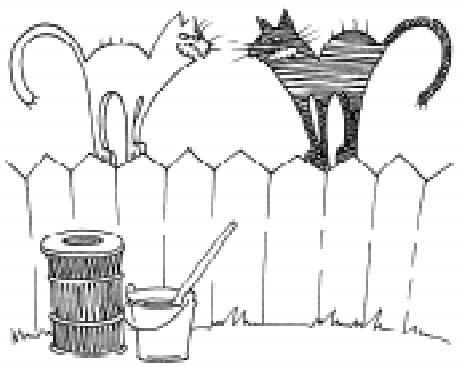
а *б*

Рис. 1



а *б*

Рис. 2



ны оба блока длиной 1, поэтому надо закрасить все оставшиеся клетки белым цветом (для обозначения белых клеток мы рисуем точку). В первом и, аналогично, последнем столбцах можно также разобраться, как расставлены блоки чёрных клеток. Чёрная клетка посредине столбца может быть только лишь частью блока длины 2, потому что иначе для этого блока не будет места. Ну а если это клетка от блока длины 2, то клетки первого и последнего столбцов могут быть раскрашены только так, как нарисовано на рис. 2*б*. Если продолжить проводить подобные рассуждения, то получится решение головоломки (рис. 1*а*).

ФОРМАЛИЗАЦИЯ ПРОЦЕССА РЕШЕНИЯ

Посмотрим, как мы решали кроссворд. На каждом шаге мы рассматривали только одну строку или только один столбец. В этой строке или столбце мы пытались найти новые клетки, цвет которых уже можно определить наверняка. В этом состоит особенность головоломки. Информация дана о строках и столбцах, но при решении трудно смотреть на подсказки сразу от нескольких строк. Видимо, так же придется работать и алгоритму автоматического решения. Алгоритм будет брать некоторую уже частично заполненную строку и искать в ней пустые клетки, цвет которых теперь можно определить. Такой алгоритм сможет решать кроссворд, просто проходясь последовательно по всем строкам и по всем столбцам, закрашивая все новые и новые клетки. Такой метод решает практически любой кроссворд из газет и журналов. Но в сложных случаях может оказаться, что кроссворд не решается только перебором строк и столбцов. Пример такого кроссворда приведен на рис 3.

Ни в какой строке и ни в каком столбце ни у какой клетки нельзя определить цвет. Как решать кроссворд в этом случае? Можно устроить перебор, что будет если закрасить верхнюю левую клетку? В этом случае можно пройтись по строкам и столбцам и убедиться, что решение придет к противоречию. Следовательно, верхнюю левую клетку надо закрасить белым цветом (поставить там точку), и после этого кроссворд можно дорешать. Предлагаем читателю проделать это самостоятельно. Заметим, что у данного кроссворда есть ровно одно решение. Любой кроссворд из газеты или журнала должен бы обладать этим свойством. Плохо если у головоломки есть несколько ответов.

Нам осталось найти алгоритм, который будет брать некоторую частично заполненную строку и искать в ней новые клетки, которые можно закрасить. Ниже мы разберём такой алгоритм, который будет работать достаточно эффективно. Значит ли это, что и решение для всего кроссворда можно найти эффективно? В статье [1], которую цитируют практически все авторы текстов о японских кроссвордах, показано, что задача «Проверить, имеет ли данный кроссворд решение», является NP-полной. Если не вдаваться в обсуждение понятия NP-полноты, то можно просто сказать, что, вероятно, не существует эффективного (полиномиального по времени) алгоритма для решения японских кроссвордов. Если такой алгоритм найдется, то можно будет эффективно (полиномиально по времени) решать любую NP-полную задачу. Среди этих задач есть задачи намного более полезные для практики, чем японские кроссворды, но найти для них полиномиальный алгоритм пока никому не удалось.

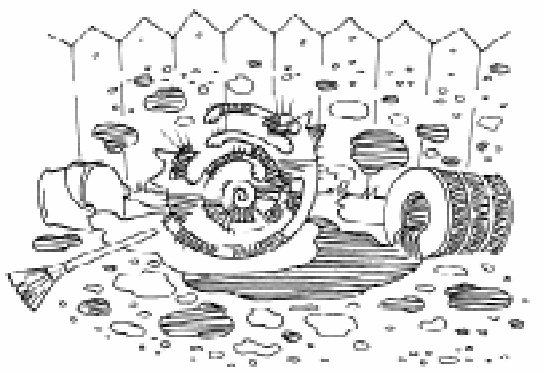
В той же статье [1] показано, что не просто проверка существования решения у кроссворда – это NP-полная задача, но NP-полной является также и проверка, есть ли

у кроссворда ещё одно решение, если какое-то уже известно. Это может быть проблемой для составителей кроссвордов, потому что хороший кроссворд должен иметь только одно решение, и это надо как-то уметь проверять.

Осталось обсудить, что на самом деле означает поиск в строке клеток, цвет которых можно определить наверняка. На рис. 4. вверху нарисована частично заполненная строка S (чёрные клетки обозначены крестиком).

При взгляде на неё понятно, например, что самая первая клетка обязательно белая, потому что никакой из двух блоков длины 3 и 2 не влезет в зазор до второй белой клетки. Сразу не определить цвета оставшихся клеток.

Нарисуем тогда все возможные варианты заполнения строки S с учётом уже заполненных клеток. Мы получим строки S_1, S_2 и S_3. Заметим, что во всех этих строках клетки 4, 5 и 8 всегда чёрные. То есть мы можем сделать вывод, что 4, 5 и 8 клетки чёрные, независимо от того, какой из вариантов S_1, S_2 или S_3 является правильным заполнением строки S. Если теперь посмотреть, например, на третью клетку, то мы видим, что в некоторых случаях она белая, в некоторых чёрная, следовательно, её цвет принципиально невозможно определить, если пользоваться только информацией, предоставленной строкой S. Итак, вся информация, которую можно получить из строки S, записана в строке S_{max}. Теперь мы знаем, что должен делать алгоритм решения строки японского кроссворда: по данной строке S ему необходимо построить строку S_{max}, добыв из S всю возможную информацию.



Человек, который решает кроссворд, за раз добывает из строки чаще всего не всю информацию, а только самую очевидную. Для этого он использует какие-нибудь эвристические приёмы, усложняющиеся по мере накопления опыта решения кроссвордов. Пример простейшего приёма решения – если строка начинается с чёрной клетки, значит, эта клетка принадлежит первому блоку строки, и первый блок можно полностью нарисовать. Программа автоматического решения для строки тоже может использовать человеческие приемы решения. Накопив достаточно много приёмов, программа будет хорошо справляться с газетными кроссвордами.

РЕШЕНИЕ СТРОКИ ЯПОНСКОГО КРОССВОРДА С ПОМОЩЬЮ ПОЛНОГО ПЕРЕБОРА

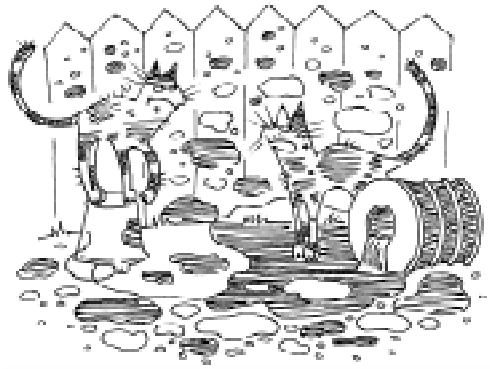
В предыдущем разделе мы сформулировали задачу. Надо создать алгоритм, который берет некоторую частично заполненную строку японского кроссворда S и находит все клетки, цвет которых можно однозначно определить. Давайте поинтересуем-

		1		1
	2	1	2	1
1	1			
1	1			
1	1			
1	1			

Рис. 3

$S =$	3	2		○			×				
$S_1 =$	3	2	○	○	×	×	×	○	×	×	○
$S_2 =$	3	2	○	○	×	×	×	○	○	×	×
$S_3 =$	3	2	○	○	○	×	×	×	○	×	×
$S_{max} =$	3	2	○	○		×	×			×	

Рис. 4



ся, что будет, если устроить полный перебор всех возможных заполнений S. То есть найти все строки S_1, S_2, S_3, ..., а потом определить клетки, которые во всех этих строках имеют один цвет. Посмотрим, сколько различных заполнений может иметь одна строка. Представим, что строка S имеет длину n, и при этом в ней не заполнена ни одна клетка. Пусть её описание – это (1, 1, 1, ...), набор из k единичек. В качестве несложной задачи по комбинаторике можно проверить, что количество возможных заполнений строки S – это C^k_{n+1-k}. При фиксированной длине строки n это выражение достигает максимума при $k \approx 0{,}28n$. (Проверьте это). Таким образом, максимальное количество заполнений строки может быть около $C^{0{,}28n}_{0{,}72n+1}$, эта функция растет быстрее любого многочлена, и в качестве задачи можно проверить, какую именно скорость роста она имеет. Вычислим значения функции для стандартных размеров кроссвордов, встречающихся в газетах. При $n = 20$ строка может быть заполнена до 3432 различными способами, при $n = 30$ получается 497420 заполнений, при $n = 50$ количество заполнений уже $6{,}1 \cdot 10^9$, при $n = 100$ – $1{,}2 \cdot 10^{20}$. Следовательно, метод полного перебора вполне работает для не очень больших кроссвордов, а для больших кроссвордов его необходимо оптимизировать.

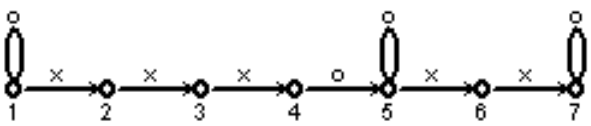


Рис. 5

РЕШЕНИЕ С ПОМОЩЬЮ КОНЕЧНОГО АВТОМАТА

Будем искать решение для строки кроссворда без полного перебора. Сначала разберёмся, как можно иначе посмотреть на набор чисел, описывающих строку. Посмотрим снова на строку S из рис. 4. Она описывается набором чисел (3, 2). Зададим *конечный автомат*, соответствующий описанию (3, 2). Он изображен на рис. 5.

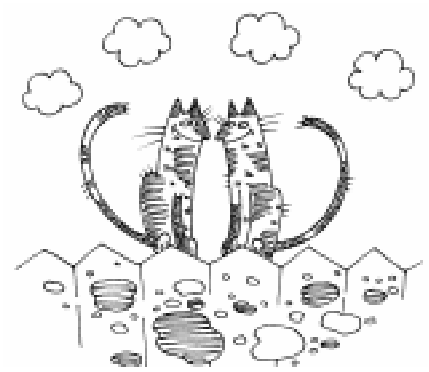
Мы не будем останавливаться на разборе понятия конечный автомат, всё равно мы достаточно подробно посмотрим на его работу. Для начала поймём, почему мы задали автомат именно таким образом. Пройдёмся по его ребрам из первой вершины в последнюю, не заходя при этом на петельки. Если выписать символы на ребрах, которые мы проходили, то получится ×××○××. Если считать, что × соответствует чёрной клетке, а ○ – белой, то получилось, что мы нарисовали строку, соответствующую описанию (3, 2). Пройдёмся ешё раз по автомату из первой вершины в последнюю, при этом пройдёмся два раза по первой петельке и один раз по последней. Получится ○○×××○××○. Это строка также соответствует описанию (3, 2) и, вообще говоря, это строка S_1 из примера на рис. 4.

Автоматом можно пользоваться не только для того, чтобы рисовать строчки с описанием (3, 2). Его можно использовать также для того, чтобы проверять, подходит ли некоторая строка под описание (3, 2). Возьмем, например, строку S_2 = ○○×××○○××. Зафиксируем свое положение в первой, начальной вершине автомата. Прочитаем первый символ S_2 – это ○. В автомате нам надо пройтись по ребру ○ – это петелька, после нее мы снова попадаем в вершину 1. Читаем следующий символ S_2 – это ○, снова проходимся по петельке и попадаем в вершину 1. Следующий символ – это ×, если пойти по ребру ×, мы попадем в вершину 2. Далее, прочитав ×, мы попадем в вершину 3, и процесс продолжится. Прочитав все символы, мы попадем в конечную вершину 7, это означает, что изучаемая строка принимается конечным автоматом, то есть соответствует описанию (3, 2).

Проверим, принимается ли автоматом строка ○○○○○○○○○. После ее прочтения, автомат окажется в вершине 1. Это значит, что строка не принялась, так как оказаться необходимо в последней 7-ой вершине. Строка ×○○○○○○○○ тоже не примется автоматом, потому что после первого шага автомат окажется во второй вершине и дальше не сможет никуда перейти, так как очередной символ – это ○, при том что такого ребра из вершины 2 нет. Заметим, что обе последние строки не подходят под описание (3, 2). Наш автомат действительно принимает те и только те строки, которые подходят под описание (3, 2).

Что, если наша строка заполнена только частично? То есть некоторые клетки в ней пустые. Как узнать, можно ли дозаполнить строку так, чтобы ее принял автомат? Другими словами, чтобы она подошла бы под описание (3, 2)? Проведем рассуждения на примере строки *S* = -○--×---- из рис. 4. Начнем принимать ее автоматом. Встанем в первую вершину автомата и прочитаем первую клетку *S*. Первая клетка неизвестна, это может быть и ×, и ○. Значит, в автомате мы можем пойти либо по ребру ×, либо по ребру ○. При этом мы попадём либо в вершину 2, либо в вершину 1. Запомним это: после первого шага мы находимся в одной из вершин множества {1, 2}. Читаем следующий символ – это ○. Из вершины 1 можно по петле перейти в вершину 1. Но из вершины 2 перейти никуда нельзя, значит, после второго шага мы находимся в вершине из множества {1}. Следующий символ неизвестен, мы попадаем в вершину из множества {1, 2}, далее символ снова неизвестен, мы попадаем в одну из вершин множества {1, 2, 3}. Читаем дальше символ ×. Если перейти по ребру с этим символом, мы можем попасть только в одну из вершин множества {2, 3, 4}.

Следующее множество возможных вершин – это {3, 4, 5}, потом {4, 5, 6}, {5, 6, 7}. Итак, после чтения всех символов строки *S* мы можем оказаться в одной из вершин множества {5, 6, 7}, то есть, в частности, можем оказаться в 7-ой конечной вершине, значит, строку *S* можно дозаполнить так,



чтобы она принялась конечным автоматом. Или, другими словами, *S* можно дозаполнить так, чтобы она стала соответствовать описанию (3, 2). Если бы после чтения последнего символа в множестве возможных вершин не было бы конечной 7-ой вершины, мы бы сделали вывод, что любое дозаполнение строки *S* не сделает ее строкой вида (3, 2).

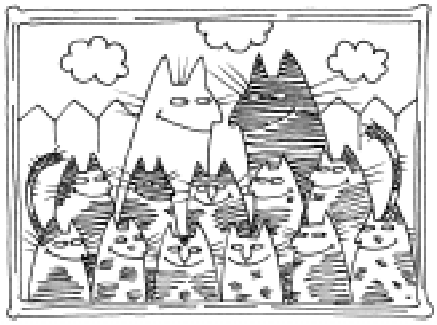
Время работы описанного алгоритма $O(n^2)$, так как необходимо прочитать n символов строки, и каждое чтение символа сопровождается изменением множества возможных вершин, размер которого ограничивается сверху n.

Мы готовы описать **первый алгоритм решения строки японского кроссворда**. Пусть дана частично заполненная строка *S* с набором описывающих чисел *def*. Алгоритм имеет следующие шаги (см. листинг 1).

Заметим, что этот алгоритм делает ровно то, что нужно: он достаёт всю возможную информацию из строки и даже умеет обнаруживать противоречие, если строку никак не получается заполнить. Время работы алгоритма $O(n^3)$, так как необходимо перебрать n клеток, а определение, можно ли клетку заполнить некоторым цветом требует $O(n^2)$ операций. При своей работе алгоритм использует память размера $O(n)$, в ней он хранит исходную строку и множества возможных вершин.

ОПТИМИЗАЦИЯ АЛГОРИТМА С КОНЕЧНЫМ АВТОМАТОМ

Приведем алгоритм решения строки японского кроссворда, который совершает всего $O(n^2)$ операций, но требует память



размера $O(n^2)$. Именно этот алгоритм содержится в демонстрационной программе из системы DADemo.

В предыдущем алгоритме мы пытались принять с помощью автомата не исходную строку *S*, а строку, получающуюся из *S* закрашиванием какой-то клетки. Нам достаточно было узнать, принимается ли измененная строка автоматом, то есть узнать, действительно ли можно закрасить клетку так, как мы её закрасили. Сейчас мы будем принимать автоматом только исходную строку *S*, но при этом нас будет интересовать не просто, принимает ли её автомат (скорее всего принимает, иначе в кроссворде есть противоречие), а то, как именно он её принимает.

Работу алгоритма продемонстрируем на примере всё той же строки *S* из рис. 4. На рис. 6 показан результат первого этапа вычислений.

Заметим, что программа изображает конечный автомат, соответствующий описанию строки. В нашем случае это описание (3, 2). Прямо под автоматом изображено исходное заполнение строки *S*. Смотрим на большую прямоугольную таблицу, её столбцы соответствуют клеткам исходной строки, они специально нарисованы под ними. Самый первый столбец будем считать нулевым, он не соответствует никакой клетке. Строки таблицы соответствуют вершинам конечного автомата.

Первый этап работы алгоритма – это попытка принять исходную строку конечным автоматом. Большая жирная точка в левом верхнем углу таблицы означает, что, до того как была прочитана какая-либо клетка, автомат находился в первой вершине. Читаем первую клетку, она не определена. Значит, мы можем считать её либо ○ – тогда

Листинг 1.

1. Взять очередную еще не заполненную клетку ***C;***
2. Получить строку S', заполнив C черным цветом;
3. Описанным только что методом проверить, дозаполняется ли S' до строки с описанием def;
4. Если ДА, то клетка C может быть черной, если НЕТ – клетка C не может быть черной;
5. Получить строку S'', заполнив C белым цветом;
6. Описанным только что методом проверить, дозаполняется ли S'' до строки с описанием def;
7. Если ДА, то клетка C может быть белой, если НЕТ – клетка C не может быть белой;
8. Определить цвет C:
   a. Если клетка C может быть белой и может быть черной, то оставить клетку C неопределенной;
   b. Если клетка C не может быть белой и может быть черной, то заполнить C черным цветом;
   c. Если клетка C может быть белой и не может быть черной, то заполнить C белым цветом;
   d. Если клетка C не может быть белой и не может быть черной, то выдать ПРОТИВОРЕЧИЕ
9. Вернуться к шагу 1;

автомат перейдет по петле и вернется в вершину 1, либо × – тогда автомат перейдёт по ребру вперед в вершину 2. В таблице это обозначено следующим образом: в первом столбце, соответствующем результату чтения первой клетки, заполнены ячейки в первой и второй строках. Что означают стрелки в таблице? Стрелка говорит о том, как мы попали в данную вершину. Например, горизонтальная стрелка с точкой в первой строке первого столбца говорит «**после чтения одной клетки можно попасть в вершину 1, перейдя из вершины 1 по ребру, помеченному точкой**». Наклонная стрелка во второй строке первого столбца говорит «**после чтения одной клетки можно попасть в вершину 2, перейдя из вершины 1 по ребру, помеченному крестиком**». Аналогично устанавливаются смыслы всех стрелок таблицы. Для ещё одного примера приведем смысл стрелок из правой нижней клетки. Они говорят «**после чтения 9 клеток можно попасть в вершину 7, перейдя из вершины 6 по ребру, помеченному крестиком, или из вершины 7 по ребру, помеченному точкой**».

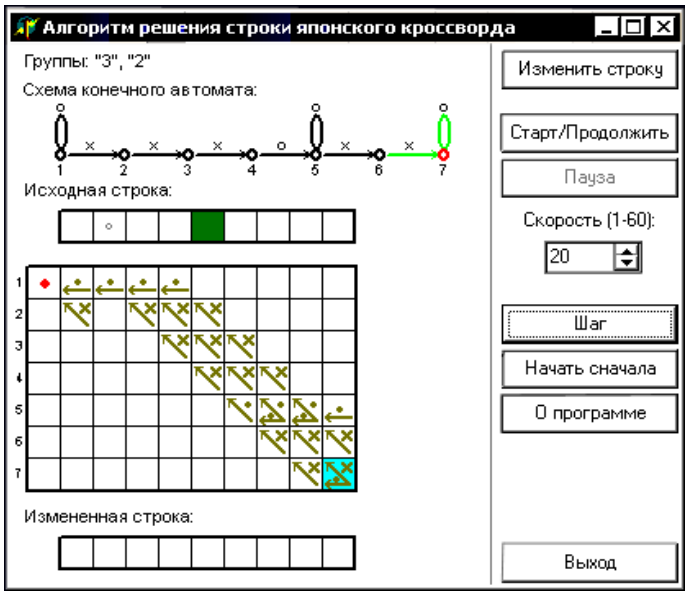


Рис. 6

Если правая нижняя клетка не заполнена, можно сделать вывод, что строка имеет противоречие. Потому что в этом случае никакое её заполнение нельзя принять конечным автоматом, то есть никакое её заполнение не имеет вид (3, 2).

Зачем мы рисовали стрелочки? Узнаем их смысл, для этого пройдемся по стрелочкам из правой нижней клетки таблицы в левую верхнюю. Есть три способа это сделать. Для примера сделаем первый шаг по стрелке с точкой, далее развилок уже не будет. Если выписывать символы над стрелками, по которым мы идем, то получится последовательность ○○×××○×××○. (Мы идем справа налево, поэтому и выписываем символы справа налево). Это S_1, одно из заполнений строки S. Далее для краткости будем называть диагональными путями те пути по стрелкам, которые идут справа снизу влево вверх.

Найдем другой диагональный путь. Если мы сначала пойдём по стрелке с крестиком, а на развилке в клетке на пятой строке седьмого столбца пойдём влево, мы по-

Рис. 7

лучим последовательность ○○×××○○××. Это S_2 – другое заполнение S. Если пройтись третьим путем, получим S_3. Итак, каждый диагональный путь соответствует какому-то заполнению строки S.

Чтобы закончить алгоритм, пройдёмся *одновременно по всем* диагональным путям. Как это происходит, изображено на рис. 7.

Клетки с темными стрелками – это те клетки, до которых мы смогли дойти из клетки справа снизу. До остальных клеток мы либо пока не дошли, либо никогда не дойдем. В данный момент обрабатывается столбец 4, он выделен более темным цветом. В этом столбце мы можем дойти, как видно, только до стрелок, помеченных ×. Это значит, что любой диагональный путь в четвертом столбце проходит через символ ×, то есть любое заполнение строки S в четвертом столбце имеет чёрную клетку. Следовательно, четвертая клетка исходной строки обязательно чёрная, и мы можем записать этот наш вывод. Внизу под окном программы нарисована ешё одна строка с ответом. Её четвертая клетка уже чёрная.

Если в каком-то столбце диагональные пути проходят и через стрелку с точкой, и через стрелку с крестиком (как, например, в столбце 6), то про клетку, соответствующую этому столбцу, ничего сказать нельзя. Действительно, видно, что клетка 6 окончательной строки не заполнена.

Аналогично, если диагональные пути в каком-то столбце проходят только по стрелкам с точками, то соответствующую клетку надо заполнить точкой. Дело в том, что эта клетка в любом заполнении строки S имеет белый цвет. Так произойдет с клетками 1 и 2.

Посмотрим ешё один пример работы алгоритма. Пусть S = ---○×-----, ее длина, как видно, 10 клеток. Описание для S будет (2, 2). Попробуем сами без использования алгоритма заполнить некоторые клетки S. Видно, что единственный × в строке должен начинать блок чёрных клеток. Все блоки строки S имеют длину 2, поэтому S можно заполнить так: S' = ---○××○---. При этом мы не знаем, какой именно из двух блоков мы нарисовали. Если реализовывать программу, которая будет решать строку так же, как человек, используя эвристические приёмы, будет полезно научить её и приёму, которым мы воспользовались сейчас. Только этот приём очень специфичен, работает исключительно в случае, когда в строке есть последовательность клеток ○× и все блоки имеют одинаковую длину. Программа, решающая кроссворды, как человек, будет состоять из множества таких странных эвристик.

А как с этим примером справится алгоритм? На рис. 8 изображён результат обоих проходов алгоритма на строке S.

Рис. 8

ЗАКЛЮЧЕНИЕ

Мы увидели, как с помощью конечных автоматов можно решать японские кроссворды. Рассмотренный алгоритм решает строку кроссворда, доставая из неё всю теоретически возможную информацию. Мы увидели, что метод решения японского кроссворда компьютером полностью отличается от метода решения человеком.

Разобранный алгоритм отлично справляется со всеми газетными и журнальными кроссвордами, но в теоретическом плане он не является идеальным.

Его можно оптимизировать, и одна из идей для этого состоит в том, что ребра автомата, помеченные крестиком, всегда идут группами. Эти группы соответствуют блокам чёрных клеток. Если автомат прошелся по первому ребру с крестиком, он обязательно пройдется и по всем оставшимся ребрам с крестиком этой группы. Если длина группы велика, например, 10, получается что автомат совершает 10 однообразных шагов подряд. Этого можно избежать, и это является идеей для проведения одной из оптимизаций.

Вместо оптимизации этого алгоритма, можно придумывать и принципиально другие идеи для решения кроссворда.

Литература

1. *N. Ueda and T. Nagao.* NP-completeness results for nonogram via parsimonious reductions. Technical Report TR96-0008, Tokyo Institute of Technology, 1996.

Посов Илья Александрович,
аспирант математико-механического факультета СПбГУ.