Ministerul Educației, Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

**Курсовая работа**

**Тема: «Анализ алгоритмов решения судоку»**

**Дисциплина: Анализ и разработка алгоритмов**

Выполнил: ст. гр. TI-197

Годунов Максим

Кишинев – 2020

**Содержание**

[1 Введение 2](#_Toc61569769)

[2 Особенности выбранного алгоритма 3](#_Toc61569770)

[3 Трассировка алгоритма 4](#_Toc61569771)

[3.1 Псевдокод алгоритма 4](#_Toc61569772)

[3.2 Трассировка алгоритма 5](#_Toc61569773)

[4 Фрагменты кода и их описание 9](#_Toc61569774)

[5 Оценка сложности алгоритма 12](#_Toc61569775)

[5.1 Сложность алгоритма 12](#_Toc61569776)

[5.2 Время выполнения 12](#_Toc61569777)

[5.3 График работы алгоритма 13](#_Toc61569778)

[6 Исходный код 14](#_Toc61569779)

[6.1 Результаты программы 15](#_Toc61569780)

[7. Вывод 16](#_Toc61569781)

[8 Библиография 17](#_Toc61569782)

# **1 Введение**

Мне, как и многим другим, нравится решать головоломки. Каждый день я стараюсь разгадывать мини-кроссворды, я играл в тетрис 1 на 1 в старшей школе, а сейчас в свободное время я обычно разгадываю судоку. В данной работе я расскажу, как решается головоломка судоку.

Вот так выглядит типичная головоломка судоку. [Смотри рис.1-2]

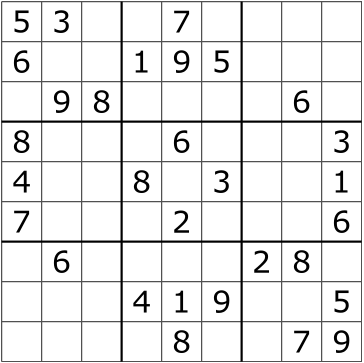
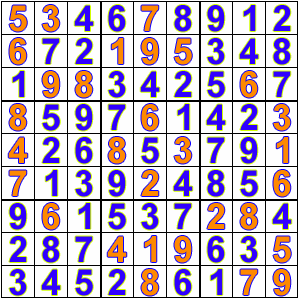
 

Рис.1(Головоломка Судоку) Рис.2(Решение)

Стандартный Судоку содержит 81 ячейку в сетке 9 × 9 и 9 прямоугольников, каждое из которых является пересечением первых, средних или последних 3 строк, а также первых, средних или последних 3 столбцов. Каждая ячейка может содержать число от одного до девяти, и каждое число может встречаться только один раз в каждой строке, столбце и поле. Судоку начинается с некоторых ячеек, содержащих числа (так называемые подсказки), и цель состоит в том, чтобы разгадать оставшиеся ячейки. У правильных судоку есть одно решение!!! Игроки и исследователи могут использовать широкий спектр компьютерных алгоритмов для решения судоку, изучения их свойств и составления новых головоломок, в том числе судоку с интересной симметрией и другими свойствами.

Существует несколько компьютерных алгоритмов, которые решают большинство головоломок 9 × 9 (n = 9) за доли секунды, но *комбинаторный взрыв1* происходит при увеличении n, ограничивая свойства судоку, которые можно построить, проанализировать и решить, как n увеличивается.

***Комбинаторный взрыв1*** — термин в математике, используемый для описания эффекта резкого («взрывного») роста временной сложности алгоритма при увеличении размера входных данных задачи.

# **2 Особенности выбранного алгоритма**

Некоторые любители разработали компьютерные программы, которые будут решать головоломки судоку с использованием алгоритма поиска с *возвратом*, который является разновидностью поиска *методом грубой силы*. Отслеживание с возвратом - это поиск в глубину (в отличие от поиска в ширину), потому что он полностью исследует одну ветвь до возможного решения, прежде чем перейти к другой ветви.

Алгоритм грубой силы посещает пустые ячейки в определенном порядке, последовательно заполняя цифры или возвращаясь, когда число оказывается недействительным. Вкратце, программа решает головоломку, помещая цифру «1» в первую ячейку и проверяя, разрешено ли ей там находится. Если нет никаких нарушений (проверка ограничений строки, столбца и поля), то алгоритм переходит к следующей ячейке и помещает в нее «1». При проверке нарушений, если обнаруживается, что «1» не допускается, значение увеличивается до «2». Если обнаруживается ячейка, в которой не допускается ни одна из 9 цифр, алгоритм оставляет эту ячейку пустой и возвращается к предыдущей ячейке. Затем значение в этой ячейке увеличивается на единицу. Это повторяется до тех пор, пока не будет обнаружено допустимое значение в последней (81-й) ячейке.

Преимущества этого метода:

* Решение гарантировано (пока головоломка действительна);
* Время решения в основном не связано со степенью сложности;
* Алгоритм (и, следовательно, программный код) проще, чем другие алгоритмы, особенно по сравнению с сильными алгоритмами, которые обеспечивают решение самых сложных головоломок;

Недостатком этого метода является то, что время решения может быть медленным по сравнению с алгоритмами, смоделированными на основе дедуктивных методов.

Таким образом, программа потратит значительное время на «подсчет» вверх, прежде чем достигнет сетки, которая удовлетворяет головоломке. В одном случае программист обнаружил, что программе грубой силы требуется шесть часов, чтобы найти решение для такого судоку (хотя и с использованием компьютера 2008 года). Такие судоку в настоящее время можно решить менее чем за 1 секунду, используя процедуру исчерпывающего поиска и более быстрые процессоры.

# **3 Трассировка алгоритма**

## **3.1 Псевдокод алгоритма**

Судоку решается путем возврата. Каждая ячейка проверяется на действительное число, перемещаясь «назад», когда есть нарушение, и снова продвигаясь вперед, пока головоломка не будет решена.

Для того, чтобы представить трассировку, ниже приведем псевдокод данного алгоритма

1. Pick empty space

2. Try all numbers

3. find that works

4. Repeat

5. If value is unavailable then backtrack

find\_empty(given\_board)

for row in given\_board:

for colum in given\_board:

if given\_board[row][colum]==0:

return (row, colum)

return None

solve(given\_board):

find = find\_empty(given\_board)

if not find:

return True

else

row, colum=find

for i 1 to 9:

if given\_board[row][colum] is valid:

given\_board[row][colum] = i

if solve(given\_board) == True

return True

given\_board[row][colum]=0

return False

## **3.2 Трассировка алгоритма**

Рассмотрим трассировку алгоритма (построчно сверху в низ)

Вставляем единицу в первую попавшуюся пустую ячейку и видим, что это значение не верно, так как у нас есть единица в строке. Следовательно, ставим следующее значение, то есть два.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** |  |  | **3** | **1** |  |  | **7** |
| **6** |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **2** |  |  | **9** |
|  | **7** |  |  | **6** |  | **2** |  |  |
|  | **9** |  |  | **4** |  | **8** |  |  |
|  | **3** |  |  |  |  | **5** |  |  |
|  |  |  |  |  |  |  |  | **1** |
|  | **2** |  |  | **9** |  |  | **8** |  |
|  |  |  |  |  |  |  | **4** |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** | **1** |  | **3** | **1** |  |  | **7** |
| **6** |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **2** |  |  | **9** |
|  | **7** |  |  | **6** |  | **2** |  |  |
|  | **9** |  |  | **4** |  | **8** |  |  |
|  | **3** |  |  |  |  | **5** |  |  |
|  |  |  |  |  |  |  |  | **1** |
|  | **2** |  |  | **9** |  |  | **8** |  |
|  |  |  |  |  |  |  | **4** |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** | **2** |  | **3** | **1** |  |  | **7** |
| **6** |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **2** |  |  | **9** |
|  | **7** |  |  | **6** |  | **2** |  |  |
|  | **9** |  |  | **4** |  | **8** |  |  |
|  | **3** |  |  |  |  | **5** |  |  |
|  |  |  |  |  |  |  |  | **1** |
|  | **2** |  |  | **9** |  |  | **8** |  |
|  |  |  |  |  |  |  | **4** |  |

Считаем относительно ячейки. В строке нет совпадений, в столбце нет совпадений, в квадрате нет совпадений, значит двойка нам подходит. Идем дальше.

В следующей пустой ячейке пробуем значения, и мы видим, что значения 1,2,3,4 нам не подходят по правилам, следовательно, вставляем 5. По тем же правилам нам в следующей пустой ячейке подходит 6. Продвигаемся дальше.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** | **2** | **5** | **3** | **1** | **6** |  | **7** |
| **6** |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **2** |  |  | **9** |
|  | **7** |  |  | **6** |  | **2** |  |  |
|  | **9** |  |  | **4** |  | **8** |  |  |
|  | **3** |  |  |  |  | **5** |  |  |
|  |  |  |  |  |  |  |  | **1** |
|  | **2** |  |  | **9** |  |  | **8** |  |
|  |  |  |  |  |  |  | **4** |  |

В данной строке у нас осталась последняя пустая ячейка и логично предположить, что сюда подходит значение 8, но если внимательно посмотреть, то в столбце мы тоже наблюдаем еще одну 8-ку, что не может являться правильным. И что мы делаем дальше?

Мы убираем 8 и отступаем назад к предыдущей ячейке и пробуем остальные значения

Значение 7 нам не подходит, значение 8 нам тоже не подходит, так как уже есть 8 в столбце. 9 нам тоже не подходит, значит отходим еще на одну ячейку назад.

Убираем 5, ставим 6 и видим, что нам это подходит.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** | **2** | **5** | **3** | **1** | **6** | **8** | **7** |
| **6** |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **2** |  |  | **9** |
|  | **7** |  |  | **6** |  | **2** |  |  |
|  | **9** |  |  | **4** |  | **8** |  |  |
|  | **3** |  |  |  |  | **5** |  |  |
|  |  |  |  |  |  |  |  | **1** |
|  | **2** |  |  | **9** |  |  | **8** |  |
|  |  |  |  |  |  |  | **4** |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** | **2** | **6** | **3** | **1** |  |  | **7** |
| **6** |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **2** |  |  | **9** |
|  | **7** |  |  | **6** |  | **2** |  |  |
|  | **9** |  |  | **4** |  | **8** |  |  |
|  | **3** |  |  |  |  | **5** |  |  |
|  |  |  |  |  |  |  |  | **1** |
|  | **2** |  |  | **9** |  |  | **8** |  |
|  |  |  |  |  |  |  | **4** |  |

Идем дальше и повторяем процесс. Видим, что из оставшихся значений (8, 9) нам ничего не подходит, значит убираем 6 и ставим 8.

В следующей ячейке нам подходит 6 В оставшейся пустой ячейке ставим последнее допустимое значение в строке, то есть 5.

И вот мы заполнили первую строку (возможно верно, так как мы не знаем, какие будут значения в остальных пустых ячейках). Дальше аналогичным способом получили первые 3 квадрата.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** | **2** | **8** | **3** | **1** | **6** | **5** | **7** |
| **6** | **1** | **3** | **5** | **7** | **9** | **4** | **2** | **8** |
| **5** | **7** | **8** | **4** | **6** | **2** | **1** | **3** | **9** |
|  | **7** |  |  | **6** |  | **2** |  |  |
|  | **9** |  |  | **4** |  | **8** |  |  |
|  | **3** |  |  |  |  | **5** |  |  |
|  |  |  |  |  |  |  |  | **1** |
|  | **2** |  |  | **9** |  |  | **8** |  |
|  |  |  |  |  |  |  | **4** |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** | **2** | **8** | **3** | **1** | **6** | **5** | **7** |
| **6** |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **2** |  |  | **9** |
|  | **7** |  |  | **6** |  | **2** |  |  |
|  | **9** |  |  | **4** |  | **8** |  |  |
|  | **3** |  |  |  |  | **5** |  |  |
|  |  |  |  |  |  |  |  | **1** |
|  | **2** |  |  | **9** |  |  | **8** |  |
|  |  |  |  |  |  |  | **4** |  |

Мы видим, что процесс трассировки вручную поэлементно по данному алгоритму занимает много времени, чтобы рассматривать детально, поэтому пропустим «парочку» шагов и получим результат

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** | **4** | **2** | **8** | **3** | **1** | **6** | **5** | **7** |
| **6** | **5** | **3** | **4** | **7** | **9** | **1** | **2** | **8** |
| **5** | **1** | **8** | **6** | **5** | **2** | **4** | **3** | **9** |
| **4** | **7** | **5** | **9** | **6** | **8** | **2** | **1** | **3** |
| **2** | **9** | **1** | **3** | **4** | **5** | **8** | **7** | **6** |
| **8** | **3** | **6** | **2** | **1** | **7** | **5** | **9** | **4** |
| **3** | **8** | **7** | **5** | **2** | **4** | **9** | **6** | **1** |
| **1** | **2** | **4** | **7** | **9** | **6** | **3** | **8** | **5** |
| **5** | **6** | **9** | **1** | **8** | **3** | **7** | **4** | **2** |

Время выполнения - 0.5252241 секунд

Время выполнения с иллюстрацией каждой итерации - 9.3156736 секунд

# **4 Фрагменты кода и их описание**

Я всегда стараюсь разбивать проблемы на мелкие кусочки, предлагая решение для них, а затем собирая все вместе. Для решения этой проблемы я начал с самых основных функций.

Просто краткое резюме: входом для этого алгоритма должен быть массив с некоторыми предварительно заполненными позициями, это будут фиксированные числа, которые нельзя изменить, как в настоящей игре судоку.

Для начала я хочу иметь функцию, которая будет печатать головоломку.

def print\_board(bo):  
 for i in range(len(bo)):  
 if i % 3 == 0 and i !=0:  
 print("- - - - - - - - - - - -")  
  
 for j in range(len(bo[0])):  
 if j % 3 == 0 and j!=0:  
 print(" | ", end="")  
  
 if j == 8:  
 print(bo[i][j])  
 else:  
 print(str(bo[i][j]) + " ", end="")

В цикле я хочу разделить доску для того, чтобы видеть различные секции. Т.е. каждый раз, когда мы на 3-ей строке, печатается «-». Каждый раз, когда мы на 3 столбце, печатается « | » и переходит к новой позиции. Дальше мы проверяем, если мы на последней позиции, мы хотим убедиться, что действительно происходит возврат и переход к следующей строке.

Дальше мы должны найти пустую ячейку, т.е. нам нужна функция, которая это выполняет

def find\_empty(bo):  
 for i in range(len(bo)):   
 for j in range(len(bo)):  
 if bo[i][j] == 0:  
 return (i,j) *#row, colum* return None

Она принимает двумерный массив (т.е. судоку) и по циклу ищет нулевые ячейки. Возвращает она индекс данной пустой ячейки. Если их нет, то ничего не возвращает

Следующим шагом мне потребуется определить, если доска с определенной позиций верна и действительна. Т.е. определяем функцию, которая принимает 3 параметра доску, число и позицию.

def valid(bo, num, position):  
 *#check row* for i in range(len(bo[0])):  
 if bo[position[0]][i] == num and position[1] != i:  
 return False  
 *#check colum* for i in range(len(bo)):  
 if bo[i][position[1]] == num and position[0] != i:  
 return False  
 *#check box* box\_x = position[1]//3  
 box\_y = position[0]//3  
 for i in range(box\_y\*3, box\_y\*3+3):  
 for j in range(box\_x\*3, box\_x\*3+3):  
 if bo[i][j] ==num and (i,j)!=position:  
 return False  
 return True

Мы должны проверить 3 вещи: строку, столбец и квадрат. В первом цикле мы перебираем каждый столбец в данной строке. Т.е мы проверяем если данной число равняется любому числу, которое мы добавили. Мы хотим убедиться, чтобы данная позиция на доске, которую мы вставляем, будет игнорироваться. Наблюдается совпадение чисел в строке, возвращаем False.

Дальше мы проверяем столбцы. Т.е. мы проходим по каждой строке от 0 до 9, и проверяем, если число в этом столбце равняется данному числу, которое мы вставляем, и также убеждаемся, что это не так же позиция для проверки, возвращаем False.

Затем мы проверяем квадраты 3 на 3, чтобы убедиться, что данное число не совпадает в квадрате. Мы определяем в каком именно квадрате находимся. Используем целочисленное деление, чтобы это выяснить. В цикле мы перебираем все 9 значений в квадрате и хотим убедиться, что не имеем такое же значение, появляющееся дважды. box\_x, box\_y получают значение либо 0, либо 1, либо 2. box\_x допустим если 2, то это означает, что мы в 3 квадрате первой строки. Т.е. мы умножаем на 3 чтобы получить индекс 6 в данном квадрате. Для box\_y происходит абсолютно также, но добавляем 3, чтобы перебрать все значения. В самом цикле мы также перебираем все значения и убеждаемся, чтобы мы не проверяли то же значение несколько раз. Возвращаем False, так как нашли такое же значение.

В конце возвращаем True, если число дойдет до конца этих проверок.

Дальше мы должны получить функцию, которая будет использовать предыдущие функции и метод возврата. Функция принимает доску и возвращает True или False

def solve(bo):  
 find = find\_empty(bo)  
 if not find:  
 return True  
 else:  
 row, col = find  
  
 for i in range(1,10):  
 if valid(bo, i, (row,col)):  
 bo[row][col] = i  
  
 if solve(bo):  
 return True  
 bo[row][col] = 0  
  
 return False

Мы будем делать это рекурсивно, что означает вызов этой же функции внутри себя. Первым делом мы определяем, если у нас нет пустых элементов, то мы нашли само решение. Иначе мы проверяем i-ое число от 1 до 9 включительно на действующую позицию. Если это число верно, то приравниваем это число к данной позиции. Затем с этой точки мы хотим проверить это решение дальше. Т.е. мы рекурсивно пытаемся закончить решение данной головоломки, вызывая данную функцию. Допустим мы вставили число 1, и дальше мы пытаемся искать решение с новым элементом, пока не найдем решение, или ни одно из чисел не подходит к данной ячейке. Если это происходит, т.е. перебирая значения и ни одно из них не подошло, мы возвращаем False, что означает неправильное решение и делаем возврат и скажем, что последний элемент, который мы добавили, обновляем, так как он не верный и снова перебираем для данной ячейки элементы. Таким образом, если мы не можем закончить решение, опираясь на элемент, который мы вставили, мы должны обновить его и попытаться использовать другое значение, повторяя весь этот процесс рекурсивно снова и снова.

# **5 Оценка сложности алгоритма**

## **5.1 Сложность алгоритма**

Метод грубой силы очень схож с алгоритмом поиска в глубину. Результатом поиска в глубину может являться, например, дерево обхода и нумерация вершин (в прямом или обратном порядке). В моем же случае результатом является заполненная матрица 2 на 2, то есть решение судоку.

Алгоритм выполняет ограниченный объём работы для каждой пустой ячейки, поэтому его сложность составляет O(n).

Так же сложность метода грубой силы зависит от размерности пространства всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение месяцев и даже лет, что собственно является недостатком

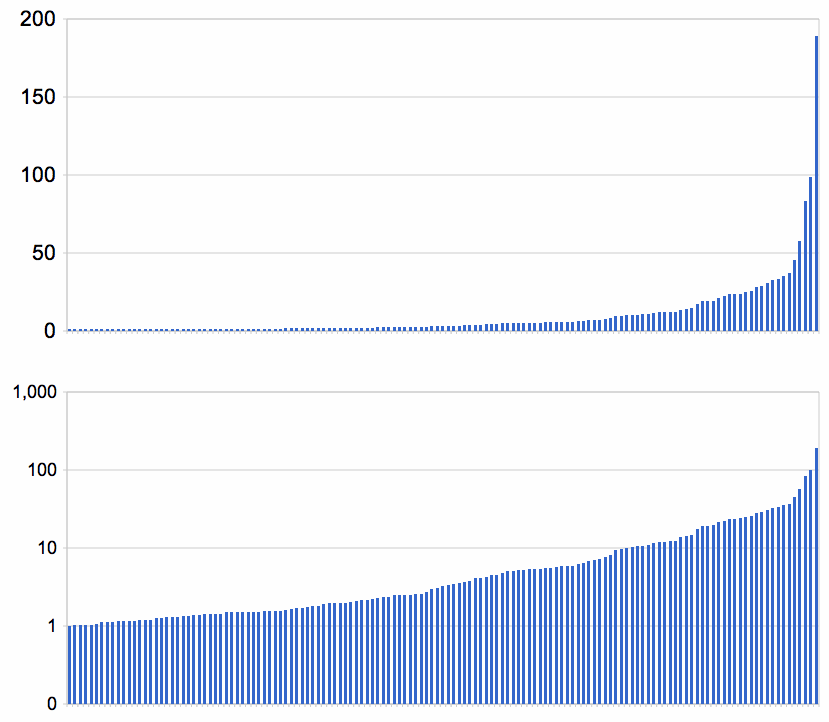
## **5.2 Время выполнения**

Для определения времени я решил проверить некоторое количество головоломок судоку. Я не умею составлять сложные головоломки, поэтому собрал около 200 случайных головоломок, сделанных самостоятельно. Исходя из этого мои случайные головоломки не дают гарантии, что есть одно уникальное решение. У многих есть несколько решений, а у некоторых (около 0,2%) нет решения.

Среднее время для решения случайной головоломки составляет 0,01 секунды, а более 99,95% занимает менее 0,1 секунды, но для некоторых требуется гораздо больше:

* 0,066% (1 из 15) заняло более 0,1 секунды
* 0,033% (1 из 30) заняло более 1 секунды
* 0,015% (1 из 65) заняло более 10 секунд
* 0,011% (1 из 90) заняло более 100 секунд

**5.3 График работы алгоритма**



Время выполнения

Время в секундах

Количество головоломок

Вот время в секундах для 200 головоломок, на выполнение которых ушло больше секунды.

# **6 Исходный код**

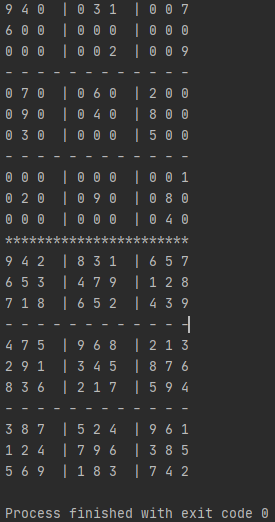
Я использовал язык программирования Python. Пустые ячейки заполнил 0.

given\_board = [  
 [9,4,0,0,3,1,0,0,7],  
 [6,0,0,0,0,0,0,0,0],  
 [0,0,0,0,0,2,0,0,9],  
 [0,7,0,0,6,0,2,0,0],  
 [0,9,0,0,4,0,8,0,0],  
 [0,3,0,0,0,0,5,0,0],  
 [0,0,0,0,0,0,0,0,1],  
 [0,2,0,0,9,0,0,8,0],  
 [0,0,0,0,0,0,0,4,0]  
]

def solve(bo):  
 find = find\_empty(bo)  
 if not find:  
 return True  
 else:  
 row, col = find  
  
 for i in range(1,10):  
 if valid(bo, i, (row,col)):  
 bo[row][col] = i  
  
 if solve(bo):  
 return True  
 bo[row][col] = 0  
  
 return False  
  
def valid(bo, num, position):  
 #check row  
 for i in range(len(bo[0])):  
 if bo[position[0]][i] == num and position[1] != i:  
 return False  
 #check colum  
 for i in range(len(bo)):  
 if bo[i][position[1]] == num and position[0] != i:  
 return False  
 #check box  
 box\_x = position[1]//3  
 box\_y = position[0]//3  
 for i in range(box\_y\*3, box\_y\*3+3):  
 for j in range(box\_x\*3, box\_x\*3+3):  
 if bo[i][j] ==num and (i,j)!=position:  
 return False  
  
 return True

def print\_board(bo):  
 for i in range(len(bo)):  
 if i % 3 == 0 and i !=0:  
 print("- - - - - - - - - - - -")  
  
 for j in range(len(bo[0])):  
 if j % 3 == 0 and j!=0:  
 print(" | ", end="")  
  
 if j == 8:  
 print(bo[i][j])  
 else:  
 print(str(bo[i][j]) + " ", end="")  
def find\_empty(bo):  
 for i in range(len(bo)):  
 for j in range(len(bo)):  
 if bo[i][j] == 0:  
 return (i,j) #row, colum  
 return None  
  
print\_board(given\_board)  
solve(given\_board)  
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
print\_board(given\_board)

## **6.1 Результаты программы**



# **7. Вывод**

В данной работе мы рассмотрели головоломку судоку. Эта головоломка по максимуму проверит силу вашего мозга. Для автоматизации решения использовался алгоритм грубой силы, который был реализован на языке Python. Такой алгоритм нельзя назвать оптимальным, но он гарантировано найдет решение головоломки, если оно есть. Конечно, если попадется довольно сложная головоломка, то я постараюсь решить ее самостоятельно, ведь никто в реальной жизни не решает эту головоломку, как показано в этой работе. Из практики выполнения данного алгоритма, среднее время решения судоку сложности чуть выше среднего составляет 0,7 секунд, а с иллюстрацией каждой итерации составляет 9,5 секунд. Я уверен, что есть и более сложные подходы в построении решения Судоку. Моя цель была достигнута, мы описали и реализовали данный алгоритм, то есть получился рабочий алгоритм. Благодаря этой работе, попутно я научился чему-то о языке Python.

# **8 Библиография**

<https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%B4%D0%BE%D0%BA%D1%83>

<https://ru.qaz.wiki/wiki/Sudoku_solving_algorithms>

<https://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html>

<https://www.mirror.co.uk/news/weird-news/worlds-hardest-sudoku-can-you-242294>

Карандашно-бумажный алгоритм решения головоломок судоку на английском:

<http://www.ams.org/notices/200904/rtx090400460p.pdf>