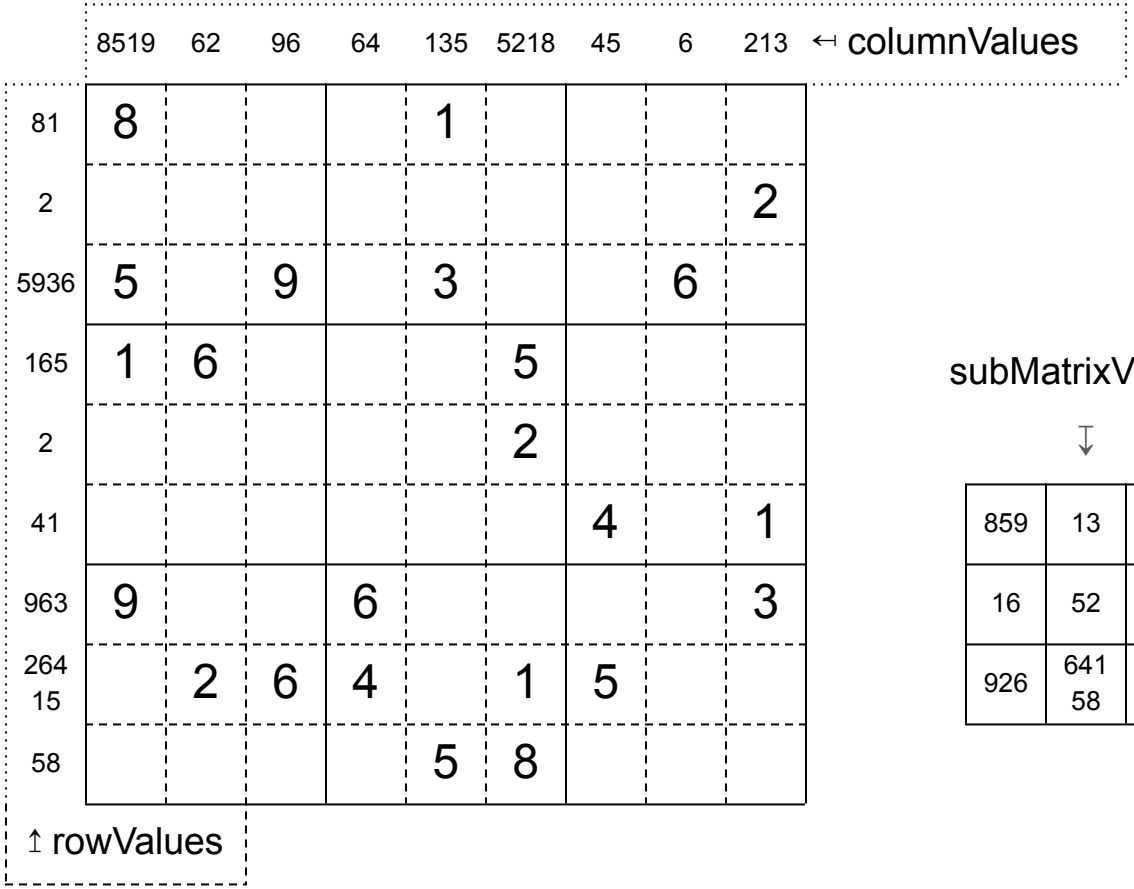
SudokuSolver

Av: Erik Cervin-Ellqvist

Ett Sudoku

8				1				
								2
5		9		3			6	
1	6				5			
					2			
						4		1
9			6					3
	2	6	4		1	5		
				5	8			

Redan existerande värden



subMatrixValues

↧

859	13	26
16	52	41
926	641 58	53

8	347	2347	2579	1	4679	379	345 79	4579
234 67	1347	1347	5789	467 89	4679	137 89	1345 789	2
5	147	9	278	3	47	178	6	478
1	6	234 78	3789	4789	5	237 89	237 89	789
347	345 789	345 78	135 789	467 89	2	367 89	357 89	567 89
237	357 89	235 78	3789	6789	3679	4	235 789	1
9	145 78	145 78	6	27	7	1278	124 78	3
37	2	6	4	79	1	5	789	789
347	1347	1347	237	5	8	126 79	124 79	4679

Möjliga värden

= checkedSingleValues

= uncheckedSingleValues

```
for columnindex, column in enumerate(tempmatrix):
    for rowindex, row in enumerate(column):
        if not len(row):
            values = []
            for value in possiblevalues:
                if value not in columnvalues[columnindex] and value not in rowvalues[rowindex] and value not in submatrixvalues[self.__tempsudoku__.getSubmMatrixIndex(columnindex, rowindex)]:
                    values.append(value)
            tempmatrix[columnindex][rowindex] = values
            if len(values) == 1:
                self.__uncheckedsinglevalues__.append(values)
self.__tempsudoku__.setMatrix(tempmatrix)
```

self.solve()

```
def solve(self):
    while not self.isSolved():
        while len(self.__uncheckedsinglevalues__):
            uncheckedvalue = self.__uncheckedsinglevalues__.pop(0)
            tempindex = self.__tempsudoku__.findIndex(uncheckedvalue)
            columnindex = tempindex[0]
            rowindex = tempindex[1]
            self.pruneValue(self.__tempsudoku__, uncheckedvalue)
            self.__checkedsinglevalues__.append(uncheckedvalue)
            self.__checkedsudoku__[columnindex][rowindex] = uncheckedvalue
```

Pruning - 1/4

8	347	2347	2579	1	4679	379	345 79	4579
234 67	1347	1347	5789	467 89	4679	137 89	1345 789	2
5	147	9	278	3	47	178	6	478
1	6	234 78	3789	4789	5	237 89	237 89	789
347	345 789	345 78	135 789	467 89	2	367 89	357 89	567 89
237	357 89	235 78	3789	6789	3679	4	235 789	1
9	145 78	145 78	6	27	7	1278	124 78	3
37	2	6	4	79	1	5	789	789
347	1347	1347	237	5	8	126 79	124 79	4679

= checkedSingleValues

= uncheckedSingleValues

= valueToPrune

= columnPrune

= rowPrune

= subMatrixPrune

```
def pruneValue(self, sudoku, valuetoprune):
    tempindex = sudoku.findIndex(valuetoprune)
    columnindex = tempindex[0]
    rowindex = tempindex[1]

    self.pruneSubMatrix(sudoku, valuetoprune, columnindex, rowindex)
    self.pruneColumn(sudoku, valuetoprune, columnindex)
    self.pruneRow(sudoku, valuetoprune, rowindex)
```

Pruning - 2/4

8	347	2347	2579	1	4679	379	345 79	4579
234 67	1347	1347	5789	467 89	4679	137 89	1345 789	2
5	147	9	278	3	47	178	6	478
1	6	234 78	3789	4789	5	237 89	237 89	789
347	345 789	345 78	135 789	467 89	2	367 89	357 89	567 89
237	357 89	235 78	3789	6789	3679	4	235 789	1
9	145 78	145 78	6	27	7	1278	124 78	3
37	2	6	4	79	1	5	789	789
347	1347	1347	237	5	8	126 79	124 79	4679

= checkedSingleValues

= uncheckedSingleValues

= valueToPrune

= removedValue

= columnPrune

= rowPrune

= subMatrixPrune

```
def pruneColumn(self, sudoku, value, columnindex):
    tempcolumn = sudoku.getColumn(columnindex)
    for values in tempcolumn:
        if value[0] in values:
            self.valueCheck(value, values)
            values.remove(value)
```

Pruning - 3/4

8	347	2347	2579	1	4679	379	345 79	4579
234 67	1347	1347	5789	467 89	4679	137 89	1345 789	2
5	147	9	278	3	47	178	6	478
1	6	234 78	3789	4789	5	237 89	237 89	789
347	345 789	345 78	135 789	467 89	2	367 89	357 89	567 89
237	357 89	235 78	3789	6789	3679	4	235 789	1
9	145 78	145 78	6	27	7	1278	124 78	3
37	2	6	4	79	1	5	789	789
347	1347	1347	237	5	8	126 79	124 79	4679

= checkedSingleValues

= uncheckedSingleValues

= valueToPrune

= removedValue

= columnPrune

= rowPrune

= subMatrixPrune

```
def assistingValueCheck(self, value1, value2):
    if len(value1) == 1 and len(value2) == 1 and value1[0] == value2[0] and value1 is not value2:
        self.__paradox__ = True

def valueCheck(self, value1, value2):
    self.assistingValueCheck(value1, value2)
    if len(value2) == 1 and value2 not in self.__checkedsinglevalues__:
        self.__uncheckedvalues__.append(value2)
```

Pruning - 4/4

8	347	2347	2579	1	469	379	345 79	4579
234 67	1347	1347	5789	467 89	469	137 89	1345 789	2
5	147	9	278	3	4	178	6	478
1	6	234 78	3789	4789	5	237 89	237 89	789
347	345 789	345 78	135 789	467 89	2	367 89	357 89	567 89
237	357 89	235 78	3789	6789	369	4	235 789	1
9	145 8	145 8	6	2	7	128	124 8	3
37	2	6	4	9	1	5	789	789
347	1347	1347	23	5	8	126 79	124 79	4679

= checkedSingleValues

= uncheckedSingleValues

= valueToPrune

= removedValue

= columnPrune

= rowPrune

= subMatrixPrune

self.isSolved()

```
def isSolved(self):
    if len(self.__checkedsinglevalues__) == self.__tempsudoku__.getSudokuize() ^ 2:
        if self.areColumnsSolved() and self.areRowsSolved() and self.areSubmatricesSolved():
            return True
    else:
        return False
```

```
def areColumnsSolved(self):
    for column in self.__tempsudoku__.getMatrix():
        possiblevalues = copy.deepcopy(self.getAllPossibleValues())
        for value in column:
            if value[0] not in possiblevalues:
                self.__paradox__ = True
                return False
            possiblevalues.remove(value[0])
        if possiblevalues:
            return False
    return True
```

Vidareutveckling

```
while len(self.__uncheckedsinglevalues__):
```

(från solve())

Vad händer när de okontrollerade enkla talen i sudokut tar slut utan att sudokut är löst?

Sökfunktion som sparar tillståndet och utforskar olika alternativa vägar, om den inte hittar en lösning längs en väg så återgår den till det sparade tillståndet.

```
self.__paradox__ = True
```

Vad gör den här variabeln?

Sätts till True när en siffra finns flera gånger i samma kolumn/rad/submatris. Kommer *alltid* att vara False så länge som indatan är korrekt och sökfunktionen ej är implementerad.

Sökfunktionen skall använda den här variabeln för att upptäcka att den nått slutet av en ogiltig gren i sökträdet.