گزارشکار پروژه دوم

نام و نام خانوادگی: سارینا همت پور

شماره دانشجویی: 9873136

توضیح مسئله

در این تمرین قصد داریم بازی جدول سودوکو را حل کنیم. در این مسئله از الگوریتم های Backtracking و Froward Checking به علاوه ی توابع هیوریستیک MVR و Degree استفاده شده است. در ادامه به توضیح این الگوریتم ها و کد های استفاده شده میپردازیم.

- کلاس Board

```
5   public class Board {
6
7       private int size;
8       private int[][] board;
9
10      public Board(int size) {
11          this.size = size;
12          board=new int[size][size];
13      }
```

این کلاس همان جدول سودوکو میباشد که شامل اندازه ی جودل و یک آرایه ی دو بعدی (خود جدول) می شود.

- کلاس SudokuSolver

```
5   public class SudokuSolver {
6
7       int count=0;
8       boolean firstRun=true;
9       private Board board;
10
11      public SudokuSolver(Board board) {
12          this.board = board;
13      }
```

این کلاس شامل تمام توابعی ست که برای حل جدول سودوکو لازم است.
برای حل این جدول از دو هیوریستیک MRV و درجه با هم دیگر استفاده میشود. به این صورت که برای انتخاب اولین متغیر جهت مقداردهی، هیورستیک درجه و برای باقی متغیر ها، هیوریستیک MRV بکار می رود. متغیر بولین fistRun هم برای مشخص کردن همین است. به این صورت که اگر الگوریتم بار اولی باشد که اجرا میشود مقدار آن true است و بعد از آن مقدار آن false میشود. پس اگر مقدار این متغیر درست باشد باید از هیوریستیک درجه و در غیر این صورت از هیوریستیک MRV استفاده کرد.

- تابع degree()

این تابع درجه را برای هر خانه از جدول محاسبه میکند و برمیگرداند.

```
167     private int degree( int row , int column)
168     {
169         int sum=0;
170         //check row
171         for (int i = 0; i < board.getSize(); i++) {
172             if (board.getBoard()[row][i]==0)
173             {
174                 sum++;
175             }
176         }
177         sum--;
178 
179         //check column
180         for (int i = 0; i < board.getSize(); i++) {
181             if (board.getBoard()[i][column]==0)
182             {
183                 sum++;
184             }
185         }
186         sum--;
187 
188         return sum;
189     }
190 
```

مقدار درجه برای هر متغیر (خانه در جدول) برابر است با تعداد متغیر های انتساب نیافته (خانه های خالی یا با مقدار صفر) در سطر و ستونی که متغیر کنونی قرار گرفته.

- تابع isValidPlacement()

چک میکند که آیا عدد ورودی میتواند در سطر و ستون داده شده قرار بگیرد یا خیر. اگر عدد در سطر یا ستون موجود باشد false و در غیر این صورت true برمیگرداند.

```
244     private boolean isValidPlacement(int number , int row , int column)
245     {
246         return !isNumberInColumn(number,column) && !isNumberInRow(number , row);
247     }
```

- تابع sizeOfDomain()

این تابع سایز دامنه یا همان تعداد مقادیری که میتوانند در آن خانه از جدول قرار بگیرند را برمیگرداند.

```
215     private int sizeOfDomain(  int row ,int column)
216     {
217         ArrayList<Integer> usedNumbers=new ArrayList<>();
218
219
220         for (int i = 0; i < board.getSize(); i++) {
221             //check row
222             if (board.getBoard()[row][i] !=0 )
223             {
224                 if (usedNumbers.indexOf(board.getBoard()[row][i])==-1)
225                 {
226                     usedNumbers.add(board.getBoard()[row][i]);
227                 }
228             }
229
230             //check column
231             if (board.getBoard()[i][column] !=0 )
232             {
233                 if (usedNumbers.indexOf(board.getBoard()[i][column])==-1)
234                 {
235                     usedNumbers.add(board.getBoard()[i][column]);
236                 }
237             }
238         }
239
240         return board.getSize()-usedNumbers.size();
241     }
```

در این تابع چک میشود که در سطر و ستون مربوطه چه اعدادی به کار رفته است. سپس از دامنه ی اصلی آنها را برمیدارد و سایز دامنه را بعد از این عملیات حذف برمیگرداند.

- تابع chooseSquare()

این تابع مسئول انتخاب متغیر بعدی برای مقدار دهی ست. همان طور که بالاتر توضیح داده شد در بار اول از هیورستیک درجه و برای دفعات بعد از هیوریستیک MRV استفاده میشود.

```
103 @      private int[] chooseSquare()
104        {
105            //using degree for the fist square
106            if (firstRun)
107            {
108                //finding maximum degree
109                int maxDegree=degree( row: 0, column: 0);
110                int maxRow=0;
111                int maxColumn=0;
112                for (int i = 0; i < board.getSize(); i++) {
113                    for (int j = 0; j < board.getSize(); j++) {
114                        if (board.getBoard()[i][j]==0)
115                        {
116                            int tempDegree=degree(i,j);
117                            if (tempDegree>maxDegree)
118                            {
119                                maxDegree=tempDegree;
120                                maxRow=i;
121                                maxColumn=j;
122                            }
123                        }
124                    }
125                }
```

در این قسمت درجه ی تمام متغیر های خالی را میشمارد و متغیر با بیشترین درجه را برمیگرداند.

```
126
127                 int[] finalSquare=new int[2];
128                 finalSquare[0]=maxRow;
129                 finalSquare[1]=maxColumn;
130                 return finalSquare;
131             }
```

برای دفعات بعدی نیز کد زیر اجرا میشود (مقدار firstRun برابراست با false)

```
136         //using mrv
137         else
138         {
139             int minSize= board.getSize()+1;
140             int minRow=0;
141             int minColumn=0;
142             for (int i = 0; i < board.getSize(); i++) {
143                 for (int j = 0; j < board.getSize(); j++) {
144                     if (board.getBoard()[i][j]==0)
145                     {
146                         int tempSize=sizeOfDomain(i , j);
147                         if (tempSize<minSize)
148                         {
149                             minSize=tempSize;
150                             minRow=i;
151                             minColumn=j;
152                         }
153                     }
154                 }
155             }
156
157             int[] finalSquare=new int[2];
158             finalSquare[0]=minRow;
159             finalSquare[1]=minColumn;
160             return finalSquare;
161         }
```

در این جا هم سایز دامنه ی متغیرها تک تک چک میشود و متغیر با کوچیک ترین دامنه برگردانده میشود (استراتژی اول-شکست)

- تابع isComplete()

کامل بودن جدول را چک میکند (شرط خاتمه ی الگوریتم)

```java
private boolean isCompleted()
{
    for (int i = 0; i < board.getSize(); i++) {
        for (int j = 0; j < board.getSize(); j++) {
            if (board.getBoard()[i][j]==0)
            {
                return false;
            }
        }
    }
    return true;
}
```

- تابع solveSudoku()

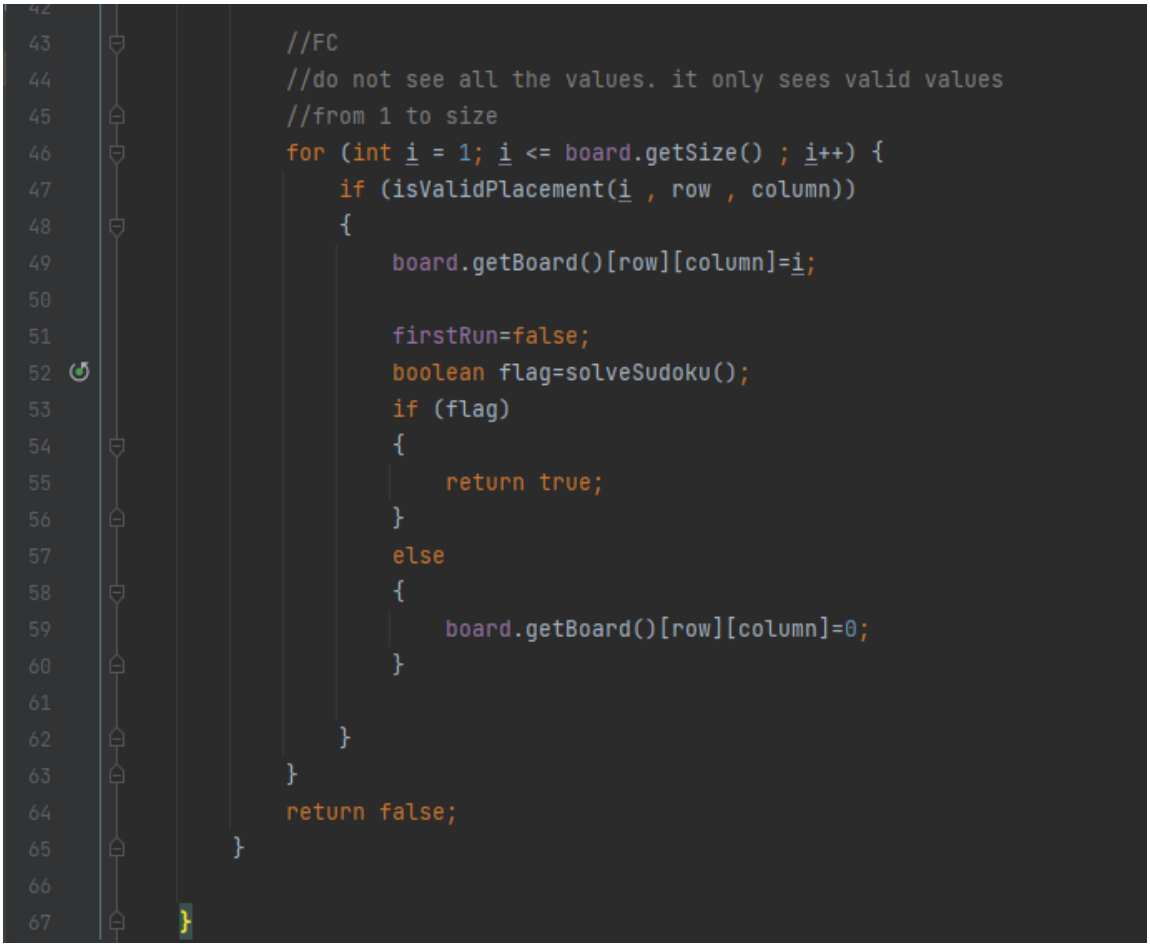
این تابع همان تابع اصلی ایست که مسئله را حل میکند. این تابع به صورت بازگشتی عمل میکند و الگوریتم Forward Checking در اینجا برای بهبود عملکر نیز بکار میرود.
Backtracking به این صورت است که بعد از انتخاب متغیر برای مقدار دهی، برای آن مقدار ست میکند و دوباره خودش را فرامیخواند. اگر در یکی از مراحل به مشکل خورد مقدار متغیر را ریست میکند (صفر قرار میدهد) و به عقب باز میگردد.
Forward Checking هم میگوید که لازم نیست تمام اعداد در متغیر قرار بگیرد وقتی میتوانیم مقادیر دامنه را فیلتر کنیم. درواقع مقادیری که مطمئن هستیم غیر مجاز هستند را از دامنه حذف میکنیم و از مقادیر باقی مانده در الگوریتم backtracking استفاده میکنیم. این کار با استفاده از تابع isValidPlacement() انجام میشود. به علاوه در ForwardChecking اگر بعد از انتساب مقدار به یک متغیر، سایز دامنه ی یکی یا بیشتر از متغیر ها صفر شد، Backtrack انجام میدهیم و ادامه نمیدهیم زیرا مطمئن هستیم که با شکست مواجه میشویم.

```
24      public boolean solveSudoku()
25      {
26          if (isCompleted())
27          {
28              return true;
29          }
30          else
31          {
32              int[] chosenSquare= chooseSquare();
33              int row=chosenSquare[0];
34              int column=chosenSquare[1];
35
36              //FC
37              // check is the size of domain is zero
38              if (sizeOfDomain(row,column)==0)
39              {
40                  return false;
41              }
```

در این کد ابتدا کامل بودن جدول را چک میکند که اگر کامل باشد یعنی باید به الگوریتم پایان دهیم و درغیر این صورت ادامه میدهیم.

در خط 32 متغیر جهت مقداردهی انتخاب میشود.

در خط 38 سایز دامنه چک میشود اگر صفر باشد یعنی مقدار متغیر قبلی که انتخاب شده اشتباه بوده. بنابراین false برمیگرداند و backtrack میکند.



```
            //FC
            //do not see all the values. it only sees valid values
            //from 1 to size
            for (int i = 1; i <= board.getSize() ; i++) {
                if (isValidPlacement(i , row , column))
                {
                    board.getBoard()[row][column]=i;

                    firstRun=false;
                    boolean flag=solveSudoku();
                    if (flag)
                    {
                        return true;
                    }
                    else
                    {
                        board.getBoard()[row][column]=0;
                    }

                }
            }
            return false;
        }

    }
```

در این قسمت هم در یک حلقه فقط مقادیر مجاز چک میشوند و به متغیر انتساب داده میشوند (الگوریتم FC)

بعد از آن در خط 52 دوباره تابع solveSudoku() فراخوانی میشود و اگر مقدار بازگشتی آن false باشد مقدار متغیر ریست میشود و یک بازگشت به مرحله ی قبل رخ میدهد. در غیر این صورت true برمیگرداند تا جایی که جدول کامل شود.

در آخر هم خروجی چاپ میشود.

پایان.