


Algorithmics	Student information	Date	Number of session
	UO: 295180	29/04/24	6
	Surname: Orviz Viesca	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Mario		



Backtracking algorithms. New approach

After revising the previous implementation of my backtracking algorithm, I realized that I was wrong in the way of solving the whole problem, as I treat rows as individual problems. The backtracking algorithm should manage the whole matrix, not only row by row.

Therefore, the previous methods solve row and solve all were deleted, and the whole matrix processing was unified in a single recursive backtracking method that is the one in charge of solving the problem.

This backtracking method is called from outside the class by its public interface that receives no parameters and performs a call to its private recursive interface, receiving an index as a parameter. This index will be in the range $[0 - \text{size}^2)$ and will be translated to cardinal coordinates and stored in a `Pair<Integer, Integer>` object by the method `translateCoords()`.

Using this, we can keep track of the index where our program is operating at each moment only by incrementing the index each time we perform a recursive call. The backtracking method performs as follows:

```
private boolean backtracking(int index) {
    Pair<Integer, Integer> coords = translateCoords(index);
    boolean isInmutable = false;
    for(int i = 0; i < 10; i++) {
        if(inmutables[coords.x][coords.y]) {
            isInmutable=true;
            break;
        }
        variableData[coords.x][coords.y] = i;
        if(coords.x==size-1) {
            if(checkCol(coords.y)) {
                if(coords.y==size-1) {
                    if(checkRow(coords.x)) {
                        return true;
                    }
                }
            }
            else {
                if(backtracking(index + 1)) return true;
            }
        }
    }
}
```

Algorithmics	Student information	Date	Number of session
	UO: 295180	29/04/24	6
	Surname: Orviz Viesca		
	Name: Mario		

```

    }
    else if(coords.y==size-1) {
        if(checkRow(coords.x)){
            if(backtracking(index + 1)) return true;
        }
    }
    else {
        if(backtracking(index + 1)) return true;
    }
}
if(isImmutable&&coords.x!=size-1&&coords.y!=size-1) {
    if(backtracking(index + 1)) return true;
}else if(isImmutable&&coords.x==size-1) {
    if(checkCol(coords.y)) {
        if(coords.y==size-1) {
            if(checkRow(coords.x)) {
                return true;
            }
        }
        else {
            if(backtracking(index + 1)) return true;
        }
    }
}else if(isImmutable&&coords.y==size-1) {
    if(checkRow(coords.x)){
        if(backtracking(index + 1)) return true;
    }
}
return false;
}

```

This method checks if the execution reaches the end of a row (pruning mechanism if the row is incorrect, we cannot follow that branch) or the end of a column (same principle).

If the algorithm reaches the right bottom corner, and the last row is solved, the problem is solved too, and the program is solved.

Note that this code only obtains 1 solution, as with the return we're breaking the execution when we find the first solution. Therefore, for the NumericSquareAll.java, some small details were modified, but the idea is still the same but without returns. The backtracking method that obtains every possible solution is a void method.

```

private void backtracking(int index) {
    Pair<Integer, Integer> coords = translateCoords(index);
    boolean isImmutable = false;
    boolean divisionRow = false;
    boolean divisionCol = false;

```

Algorithmics	Student information	Date	Number of session
	UO: 295180	29/04/24	6
	Surname: Orviz Viesca		
	Name: Mario		

```

for(int i = 0; i < 10; i++) {
    if(inmutables[coords.x][coords.y]) {
        isImmutable=true;
        break;
    }
    variableData[coords.x][coords.y] = i;
    if(coords.x>0&&colOperators[coords.x-1][coords.y]=="/") {
        if(performOperation(variableData[coords.x-1][coords.y], "/",
i)==Integer.MAX_VALUE) divisionCol=true;
        else divisionCol=false;
    }if(coords.y>0&&rowOperators[coords.x][coords.y-1]=="/") {
        if(performOperation(variableData[coords.x][coords.y-1], "/",
i)==Integer.MAX_VALUE) divisionRow=true;
        else divisionRow=false;
    }

    if(coords.x==size-1&&!divisionRow&&!divisionCol) {
        if(checkCol(coords.y)) {
            if(coords.y==size-1) {
                if(checkRow(coords.x)) {
                    numberOfSolutions++;
                    System.out.println("Solution " + numberOfSolutions);
                    printSolution();
                }
            }
            else {
                backtracking(index + 1);
            }
        }
    }
    else if(coords.y==size-1&&!divisionRow&&!divisionCol) {
        if(checkRow(coords.x)){
            backtracking(index + 1);
        }
    }
    else if(!divisionRow&&!divisionCol){
        backtracking(index + 1);
    }
}

if(coords.x>0&&colOperators[coords.x-1][coords.y]=="/") {
    if(performOperation(variableData[coords.x-1][coords.y], "/",
variableData[coords.x][coords.y])!=Integer.MAX_VALUE) divisionCol=true;
    else divisionCol=false;
}if(coords.y>0&&rowOperators[coords.x][coords.y-1]=="/") {
    if(performOperation(variableData[coords.x][coords.y-1], "/",
variableData[coords.x][coords.y])!=Integer.MAX_VALUE) divisionRow=true;
    else divisionRow=false;
}
if(isImmutable&&coords.x!=size-1&&coords.y!=size-
1&&!divisionRow&&!divisionCol) {
    backtracking(index + 1);
}
else if(isImmutable&&coords.x==size-1&&!divisionRow&&!divisionCol) {
    if(checkCol(coords.y)) {

```

Algorithmics	Student information	Date	Number of session
	UO: 295180	29/04/24	6
	Surname: Orviz Viesca		
	Name: Mario		

```

        if(coords.y==size-1) {
            if(checkRow(coords.x)) {
                numberOfSolutions++;
                System.out.println("Solution " + numberOfSolutions);
                printSolution();
            }
        }
        else {
            backtracking(index + 1);
        }
    }
}
else if(isInmutable&&coords.y==size-1&&!divisionRow&&!divisionCol) {
    if(checkRow(coords.x)){
        backtracking(index + 1);
    }
}
}
}

```

In this case, I wanted to introduce a new pruning mechanism, whenever the program finds a division operation, it checks that is a valid division. I thought that this will increase the performance of the algorithms but the truth is that it doesn't really make a great change, so I didn't implement it in the branch and bound method or in the first backtracking method, as the code is pretty complex and a mess.

<i>Test case</i>	<i>Time for first solution [ms]</i>	<i>Time for all the solutions [ms]</i>	<i>Number of solutions found</i>
<i>Test00</i>	LoR	LoR	1
<i>Test01</i>	LoR	LoR	12
<i>Test02</i>	LoR	LoR	1
<i>Test03</i>	12.6	49	3
<i>Test04</i>	14.9	LoR	2
<i>Test05</i>	16.8	308	5
<i>Test06</i>	LoR	279	83
<i>Test07</i>	50.2	OoT	+200,000