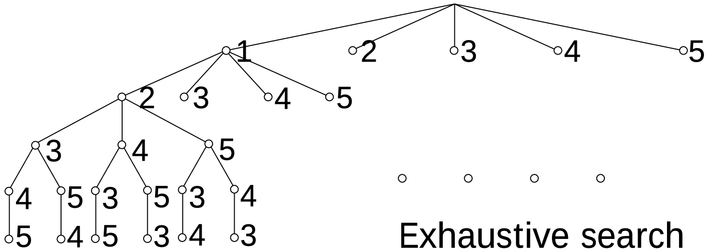
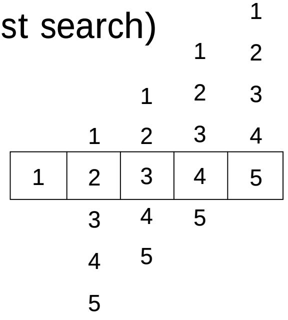
Sudoku, Mathdoku, and Related Problems

Pei-yih Ting

Generating Permutations



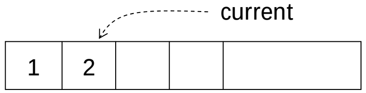
Exhaustive search
(depth first search)



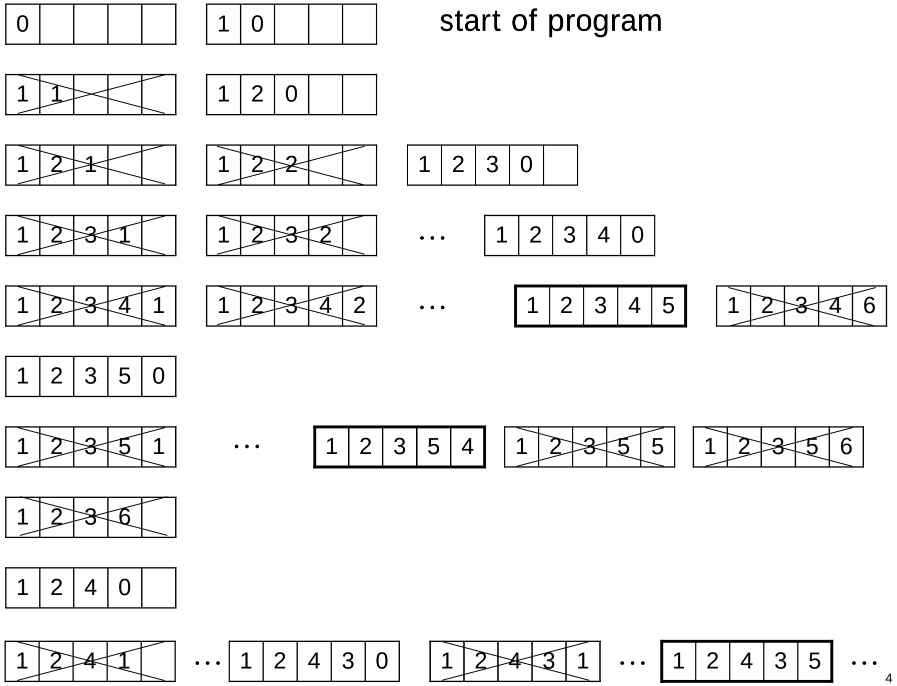
5! = 120
permutations

1 2 3 4 5
1 2 3 5 4
1 2 4 3 5
1 2 4 5 3
1 2 5 3 4
1 2 5 4 3
…

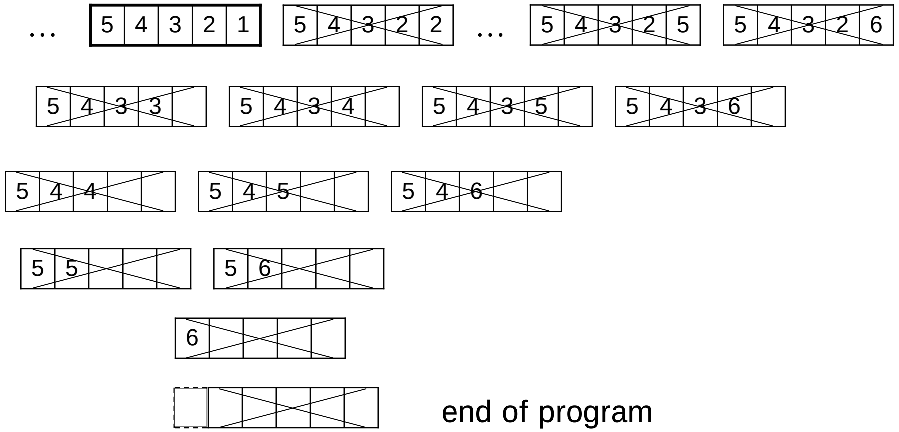
```
#include <stdio.h>

void main()
{
    int size, perm[12] = {0}, current=0, solCount=0, i;

    printf("Please input number of elements: ");
    scanf("%d", &size);

    while (current>=0)
    {
        current += next(size, current, perm);
        if (current == size)
        {
            solCount++;
            printf("%4d: ", solCount);
            for (i=0; i<size; i++)
                printf("%d ", perm[i]);
            printf("\n");
            current = size-1;
        }
    }
    printf("Total %d permutations\n",solCount);
}
```



```
int next(int size, int pivot, int perm[]
{
    int i, collision;

    while (perm[pivot]++ < size)
    {
        collision = 0;
        for (i=0; i<pivot; i++)
            if (perm[pivot] == perm[i])
            {
                collision = 1;
                break;
            }
        if (!collision) return 1;
    }
    perm[pivot] = 0;
    return -1;
}
```



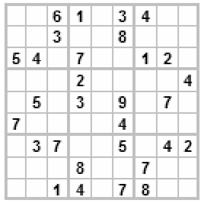
start of program



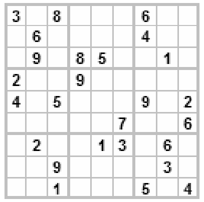
end of program

Sudoku

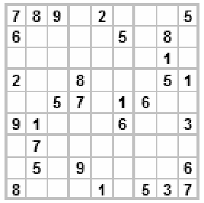
- Sudoku: In these three examples, 81 cells are divided into 9 blocks each with 9 cells (3-by-3). A player is required to fill in the blank cells such that integers in each row, each column, and each block are permutations of {1,2,3,…,9}, i.e. no duplication of numbers in each row, column, or block.



		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
	5		3		9		7	
7					4			
	3	7			5		4	2
			8			7		
		1	4		7	8		



3		8				6		
	6					4		
	9		8	5			1	
2			9					
4		5				9		2
					7			6
	2			1	3		6	
		9					3	
		1				5		4



7	8	9		2				5
6					5		8	
							1	
2			8				5	1
		5	7		1	6		
9	1				6			3
	7							
	5		9					6
8				1		5	3	7

number of lines
row, column, value
row, column, value
…

Initial configuration

30
0, 0, 7
0, 1, 8
0, 2, 9
0, 4, 2
0, 8, 5
1, 0, 6
1, 5, 5
1, 7, 8
…

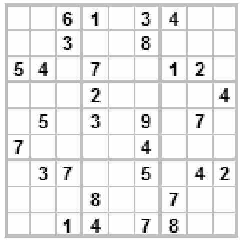
Data Representation

- Two dimensional integer array: int board[9][9]; initialized with 0's and fixed constraints

0	0	**6**	**1**	0	**3**	**4**	0	0
0	0	**3**	0	0	**8**	0	0	0
5	**4**	0	**7**	0	0	**1**	**2**	0
0	0	0	**2**	0	0	0	0	**4**
0	**5**	0	**3**	0	**9**	0	**7**	0
7	0	0	0	0	**4**	0	0	0
0	**3**	**7**	0	0	**5**	0	**4**	**2**
0	0	0	**8**	0	0	**7**	0	0
0	0	**1**	**4**	0	**7**	**8**	0	0

Depth First Search Process

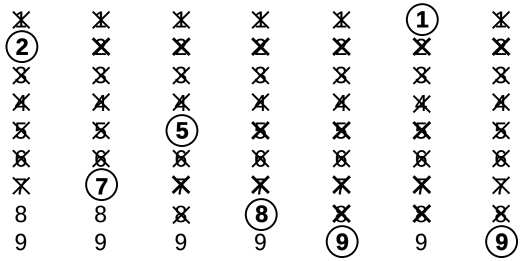
- Extension of permutation generation: more constraints on the set of numbers to be filled in each cell



		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
	5		3		9		7	
7					4			
	3	7			5		4	2
			8			7		
		1	4		7	8		

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

…



Satisfying Constraints

- For each cell
  - No two cells are assigned the same value in each row

```
for (i=0; i<9; i++)
   if (value == board[i][icol]) return 0;
```

  - No two cells are assigned the same value in each column

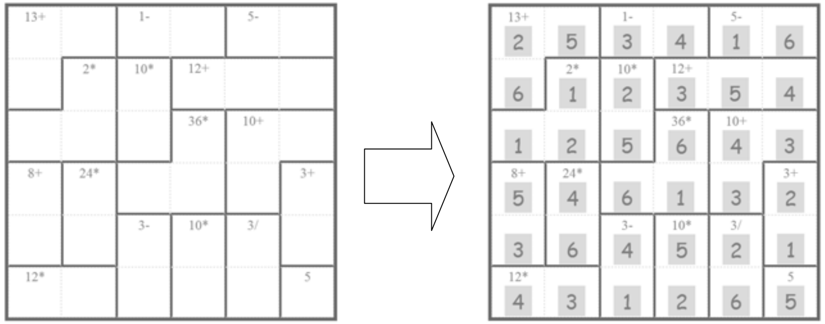
```
for (i=0; i<9; i++)
   if (value == board[irow][i]) return 0;
```

  - No two cells are assigned the same value in each 3x3 subblock

```
for (i=irow/3*3; i<irow/3*3+3; i++)
   for (j=icol/3*3; j<icol/3*3+3; j++)
      if (value == board[i][j]) return 0;
```

Mathdoku

- 4x4, 6x6, 8x8, …, nxn
- The values in each cell are from the set {1, 2, …, n}
- No repetition is allowed in each row and each column
- In addition, the values in each non-regular subblocks should satisfy the labeled arithmetic constraints, e.g.
  13+ is satisfied by 2+5+6=13, 1- is satisfied by 4-3=1,
  36* is satisfied by 6*6*1=36, and 3/ is satisfied by 6/2=3



13+		1-		5-	
	2*	10*	12+		
			36*	10+	
8+	24*				3+
		3-	10*	3/	
12*					5

13+ 2	5	1- 3	4	5- 1	6
6	2* 1	10* 2	12+ 3	5	4
1	2	5	36* 6	10+ 4	3
8+ 5	24* 4	6	1	3	3+ 2
3	6	3- 4	10* 5	3/ 2	1
12* 4	3	1	2	6	5 5

Data Representation

1. Two dimensional integer array to store the chosen numbers.
2. Two dimensional integer array to store the constraint subblocks.

For example,

```
6
+  13     3  1 2 7
-   1     2  3 4
-   5     2  5 6
*    2    3  8 13 14
*   10    2  9 15
...
$
```

0	1	0	2	0	5
2	0	0	0	10	11
8	13	9	0	0	17
0	0	16	21	18	0
19	20	0	0	0	24
0	31	27	28	29	0

+		-		-	
	*	*	+		
			*	+	
+	*				+
		-	*	/	
*					5

13		1		5	
	2	10	12		
			36	10	
8	24				3
		3	10	3	
12					5

0	0	0	**2**01	0	**2**05
113	0	0	0	0	**1**12
0	**3**02	**3**10	0	0	0
0	0	0	**3**36	**1**10	0
108	**3**24	0	0	0	**1**03
0	**3**12	**2**03	**3**10	**4**03	5

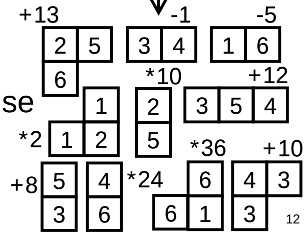
Constraints are checked at the end of a subblock

Basic Algorithm

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

- Enumerating all cells with {1, 2, …, n} starting from the cell in the left-top corner, from top to down, from left to right, i.e.
- Satisfying row and column uniqueness constraints
- Skipping all cells with fixed numbers
- For the last cell of each arithmetic constraint subblock, numbers in this subblock must satisfy the specified constraint.
- Succeeds if all cells are filled; fails otherwise
- Repeat enumerating for next possible solutions

2	5	3	4	1	6
6	1	2	3	5	4
1	2	5	6	4	3
5	4	6	1	3	2
3	6	4	5	2	1
4	3	1	2	6	5


+13
2 5
6
-1
3 4
-5
1 6
*10
1
2
5
+12
3 5 4
*2
1 2
*36
6
1
+10
4 3
3
+8
5
3
4
6
*24
6