

**BỘ MÔN KHOA HỌC MÁY TÍNH**  
**CT332: TRÍ TUỆ NHÂN TẠO**  
**BÀI THỰC HÀNH SỐ 3**  
**TÌM KIẾM THỎA MÃN RÀNG BUỘC**

**I. Mục tiêu**

- ✓ Biểu diễn bài toán trên không gian trạng thái và áp dụng thỏa mãn ràng buộc để giải quyết bài toán Sudoku.
- ✓ Ngôn ngữ sử dụng: C.

**II. Nội dung**

**1. Mô tả bài toán**

Cho một bảng số  $9 \times 9$ , trong đó có những con số được cho trước, còn lại là những ô trống. Mục tiêu là điền hết tất cả những ô trống đó sao cho mỗi hàng, mỗi cột, và mỗi khối (vùng  $3 \times 3$  được chia trước) đều chứa tất cả các chữ số từ 1 tới 9. Như vậy ta có các ràng buộc sau: chỉ điền các số từ 1 đến 9, các số trong cùng hàng không được trùng nhau, các số trong cùng cột không được trùng nhau và các số trong cùng khối không được trùng nhau.

	Ví dụ về bảng Sudoku ban đầu bao gồm: <ul style="list-style-type: none"><li>➤ Các số được cho trước thỏa mãn các ràng buộc.</li><li>➤ Các ô còn trống cần người chơi hoàn thành.</li><li>➤ Mỗi ô có miền giá trị từ 1 đến 9.</li></ul>
--	--

*Lưu ý: Trạng thái hợp lệ với một ràng buộc không có nghĩa đây là trạng thái hợp lệ với toàn bộ bài toán. Tuy nhiên, trạng thái hợp lệ với bài toán thì nó cần hợp lệ với toàn bộ các ràng buộc.*

## Buổi 3 - Tìm kiếm thỏa mãn ràng buộc

---

**Ràng buộc 1:** Mỗi ô có miền giá trị chỉ từ 1 đến 9.

5	3		7					
6			1	9	5			
	9	8		10		6		
8			6			3		
4			8	3			1	
7			2			6		
	6			2	8			
		4	1	9			5	
		8			7	9		

Trạng thái không hợp lệ vì điền số 10 (trái với ràng buộc) vào ô trống.

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

Trạng thái hợp lệ với ràng buộc 1 vì điền số 4 (thỏa mãn ràng buộc) vào ô trống.

**Ràng buộc 2:** Các số trong cùng hàng không được trùng nhau.

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

Hàng không hợp lệ vì có hai số 6 trong cùng hàng.

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

Hàng hợp lệ vì không có số nào trùng nhau trong cùng hàng và đầy đủ số từ 1 đến 9.

**Ràng buộc 3:** Những số trong cùng một cột không được trùng nhau

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

Cột không hợp lệ vì có hai số 2 trong cùng cột.

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

Cột hợp lệ vì không có số nào trùng nhau trong cùng cột và đầy đủ số từ 1 đến 9.

**Ràng buộc 4:** Những số trong cùng một khối không được trùng nhau

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

Khối không hợp lệ vì có hai số 5 trong cùng khối.

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

Khối hợp lệ vì không có số nào trùng nhau trong cùng khối và đầy đủ số từ 1 đến 9.

- Ví dụ về trạng thái hoàn thành:**

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

Bài toán được giải quyết nếu:

- Đã điền xong hết các ô trống.
- Không có ràng buộc nào bị vi phạm.

## 2. Phân tích bài toán

- Bài toán sử dụng phương pháp biểu diễn có cấu trúc để biểu diễn các trạng thái, mỗi trạng thái là một tập biến, mỗi biến có một giá trị.
- Gồm 3 thành phần: X, D và C trong đó:
  - Biến (X): tập hợp các ô trống trong bảng, còn các số được đề bài cho trước là hằng số được khởi tạo trước.
  - Miền giá trị (D): {1, 2, ..., 9}
  - Ràng buộc (C): mỗi ô chỉ có thể có các số từ 1 đến 9, các số trong cùng hàng không được trùng nhau, các số trong cùng cột không được trùng nhau và các số trong cùng khối không được trùng nhau.
- Giải quyết bài toán bằng cách sử dụng lan truyền ràng buộc
  - Dùng các ràng buộc để giảm giá trị hợp lệ cho một biến.
  - Kết quả các giá trị này có thể làm giảm các giá trị hợp lệ cho biến khác.

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

- Miền D của các ô tô đậm là các số tương ứng.
- Ở ô chính giữa đang xét có miền D = {5}, vì chỉ có duy nhất 1 lựa chọn nên ta điền 5 vào ô đó.
- Sau đó, xóa số 5 khỏi miền giá trị ở các ô bị ảnh hưởng bởi ràng buộc (ngang, dọc, khối).

### III. Cài đặt

#### 1: Cấu trúc tọa độ:

Vị trí của một ô được xác định bởi tọa độ của nó

```
typedef struct {  
    int x, y;  
} Coord;  
  
// x và y có giá trị  
từ 0 đến 8
```

Ví dụ: Ô đang xét có tọa độ (1, 4)

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

#### 2: Cấu trúc danh sách tọa độ:

Một cấu trúc List dùng để lưu trữ Coord (về cơ bản giống cấu trúc List lưu trữ số mà chúng ta đã biết)

```
typedef struct {  
    Coord data[MAX_LENGTH];  
    int size;  
} ListCoord;  
  
void initListCoord(ListCoord *list) {  
    list->size = 0;  
}  
  
void appendListCoord(ListCoord *list, Coord coord) {  
    list->data[list->size++] = coord;  
}
```

### 3: Cấu trúc ràng buộc

Các ràng buộc được biểu diễn bằng ma trận đỉnh đỉnh, thứ tự đỉnh được đánh số từ 0 đến 80 ứng với 81 ô tương ứng trong bàn cờ Sudoku theo thứ tự lần lượt từ trái qua phải, từ trên xuống dưới.

```
#define NB_ROWS 9
#define NB_COLUMNS 9

typedef struct {
    int data[NB_ROWS*NB_COLUMNS] [NB_ROWS*NB_COLUMNS];
    int n;
} Constrains;
```

5 0	3 1	2	3	7 4	5	6	7	8
6 9	10	11	1 12	9 13	5 14	15	16	17
18	9 19	8 20	21	22	23	24	6 25	26
8 27	28	29	30	6 31	32	33	34	3 35
4 36	37	38	8 39	40	3 41	42	43	1 44
45	46	47	48	2 49	50	51	52	6 53
54	6 55	56	57	58	59	2 60	8 61	62
63	64	65	4 66	1 67	9 68	69	70	5 71
72	73	74	75	8 76	77	78	7 79	9 80

- Ví dụ đỉnh 0 và đỉnh 1 có sự ràng buộc với nhau do đó :  
data[0][1] = 1
- Trong khi đó đỉnh 0 và đỉnh 12 không có sự ràng buộc:  
data[0][12] = 0
- Trong bàn cờ Sudoku, luôn có 81 ô vuông do đó số lượng đỉnh n là hằng số và bằng 81.

#### 3.1 Khởi tạo ràng buộc rỗng:

Các đỉnh lúc này chưa có sự ràng buộc

```
void initConstrains(Constrains* constrains) {
    int i,j;
    for (i = 0; i < NB_ROWS*NB_COLUMNS; i++)
        for (j = 0; j < NB_ROWS*NB_COLUMNS; j++)
            constrains->data[i][j] = 0;
    constrains->n = NB_ROWS*NB_COLUMNS;
}
```

### 3.2 Chuyển đổi giá trị :

Chuyển đổi từ tọa độ x y sang tọa độ đỉnh và ngược lại.

5 0	3 1	2	3	7 4	5	6	7	8
6 9	10	11	1 12	9 13	5 14	15	16	17
18	9 19	8 20	21	22	23	24	6 25	26
8 27	28	29	30	6 31	32	33	34	3 35
4 36	37	38	8 39	40	3 41	42	43	1 44
45	46	47	48	2 49	50	51	52	6 53
54	6 55	56	57	58	59	2 60	8 61	62
63	64	65	4 66	1 67	9 68	69	70	5 71
72	73	74	75	8 76	77	78	7 79	9 80

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

Hình trên cho thấy, đỉnh số 13 tương ứng với tọa độ (1, 4) .

+ *Tìm chỉ số đỉnh khi biết tọa độ*

*Một ô trong Sudoku chuyển thành một đỉnh trong đồ thị ràng buộc*

```
int indexOf(Coord coord) {
    return (NB_ROWS*coord.x + coord.y);
}
```

+ *Tìm tọa độ khi biết chỉ số đỉnh*

*Một đỉnh trong đồ thị ràng buộc chuyển thành một ô trong Sudoku*

```
Coord positionOfVertex(int vertex) {
    Coord coord;
    coord.x = vertex / NB_ROWS;
    coord.y = vertex % NB_COLUMNS;
    return coord;
}
```

### 3.3 Thêm ràng buộc

Trả về 1 nếu thêm thành công, 0 nếu thất bại.

```
int addConstrain(Constrains* constrains, Coord source, Coord target) {
    int u = indexOf(source);
    int v = indexOf(target);
    if (constrains->data[u][v] == 0) {
        constrains->data[u][v] = 1;
        constrains->data[v][u] = 1;
        return 1;
    }
    return 0;
}
```

### 3.4 Tập ràng buộc của một đỉnh

Để tính miền giá trị của một đỉnh, ta cần phải xác định những đỉnh nào được ràng buộc với nó.

```
ListCoord getConstrains(Constrains
constrains, Coord coord) {
    int i;
    int v = indexOf(coord);
    ListCoord result;
    initListCoord(&result);
    for (i = 0; i < constrains.n; i++) {
        if (constrains.data[v][i] == 1) {
            appendListCoord(&result,
positionOfVertex(i));
        }
    }
    return result;
}
```

5 0	3 1	2	3	7 4	5	6	7	8
6 3	10	11	1 12	9 13	5 14	15	16	17
18	9 19	8 20	21	22	23	24	6 25	26
8 27	28	29	30	6 31	32	33	34	3 35
4 36	37	38	8 39	40	3 41	42	43	1 44
45	46	47	48	2 49	50	51	52	6 53
54	6 55	56	57	58	59	2 60	8 61	62
63	64	65	4 66	1 67	9 68	69	70	5 71
72	73	74	75	8 76	77	78	7 79	9 80

Ví dụ những đỉnh được ràng buộc với đỉnh 13 bao gồm (ngang, dọc, khối):

{ 3, 4, 5, 9, 10, 11, 12, 14, 15, 16, 17, 21, 22, 23, 31, 40, 49, 58, 67, 76 }

#### 4: Cấu trúc bảng Sudoku

Bao gồm các ô và các ràng buộc.

```
#include <stdlib.h>
#define MAX_VALUE 10
#define EMPTY 0
#define AREA_SIZE 3
#define INF 999999

typedef struct {
    int cells[NB_ROWS][NB_COLUMNS];
    Constrains constrains;
} Sudoku;
```

##### 4.1: Khởi tạo bảng Sudoku rỗng

```
void initSudoku(Sudoku* sudoku) {
    int i, j;
    for (i = 0; i < NB_ROWS; i++) {
        for (j = 0; j < NB_COLUMNS; j++) {
            sudoku->cells[i][j] = EMPTY;
        }
    }
    initConstrains(&sudoku->constrains);
}
```

##### 4.2: Khởi tạo bảng Sudoku có giá trị

```
void initSudokuWithValues(Sudoku* sudoku, int
inputs[NB_ROWS][NB_COLUMNS]) {
    int i, j;
    for (i = 0; i < NB_ROWS; i++) {
        for (j = 0; j < NB_COLUMNS; j++) {
            sudoku->cells[i][j] = inputs[i][j];
        }
    }
    initConstrains(&sudoku->constrains);
}
```

#### 4.3: In bảng Sudoku

```
void printSudoku(Sudoku sudoku) {  
    int i, j;  
    printf("Sudoku:\n");  
    for (i = 0; i < NB_ROWS; i++) {  
        if (i % AREA_SIZE == 0) printf("-----\n");  
        for (j = 0; j < NB_COLUMNS; j++) {  
            if (j % AREA_SIZE == 0) printf ("| ");  
            printf ("%d ", sudoku.cells[i][j]);  
        }  
        printf ("|\n");  
    }  
    printf("-----\n");  
}
```

#### 4.4: Kiểm tra trạng thái kết thúc

Kiểm tra xem có điền hết tất cả các ô thỏa mãn các ràng buộc hay chưa

```
int isFilledSudoku(Sudoku sudoku) {  
    int i, j;  
    for (i = 0; i < NB_ROWS; i++) {  
        for (j = 0; j < NB_COLUMNS; j++) {  
            if (sudoku.cells[i][j] == EMPTY) return 0;  
        }  
    }  
    return 1;  
}
```

### 4.5: Tao ràng buộc từ vị trí cho trước

```
void spreadConstrainsFrom(Coord position, Constrains* constrains,
ListCoord* changeds) {
    int row = position.x, column = position.y;
    int i,j;
    // Tao rang buoc theo cot
    for (i = 0; i < NB_ROWS; i++) {
        if (i != row) {
            Coord pos = {i, column};
            if (addConstrain(constrains, position, pos))
appendListCoord(changeds, pos);
        }
    }
    // Tao rang buoc theo hang
    for (i = 0; i < NB_COLUMNS; i++) {
        if (i != column) {
            Coord pos = {row, i};
            if (addConstrain(constrains, position, pos))
appendListCoord(changeds, pos);
        }
    }
    // Truyen rang buoc theo khoi
    for (i = 0; i < AREA_SQUARE_SIZE; i++) {
        for ( j = 0; j < AREA_SQUARE_SIZE; j++) {
            int areaX = (row/AREA_SQUARE_SIZE) * AREA_SQUARE_SIZE;
            int areaY = (column/AREA_SQUARE_SIZE) * AREA_SQUARE_SIZE;
            if (areaX+i != row || areaY+j != column) {
                Coord pos = {areaX+i, areaY+j};
                if (addConstrain(constrains, position, pos))
appendListCoord(changeds, pos);
            }
        }
    }
}
```

#### 4.6: Tìm miền giá trị của ô trống

```
List getAvailableValues(Coord position, Sudoku sudoku) {  
    ListCoord posList = getConstraints(sudoku.constraints, position);  
    int availables[MAX_VALUE]; // 0->9 array, just use 1->9  
    int i;  
    for (i = 1; i < MAX_VALUE; i++) availables[i] = 1;  
    for (i = 0; i < posList.size; i++) {  
        Coord pos = posList.data[i];  
        if (sudoku.cells[pos.x][pos.y] != EMPTY) {//đang mang giá trị  
            availables[sudoku.cells[pos.x][pos.y]] = 0;  
        }  
    }  
    List result;  
    initList(&result);  
    for (i = 1; i < MAX_VALUE; i++) {  
        if (availables[i]) appendList(&result, i);  
    }  
    return result;  
}
```

**Chú ý:** Nếu `availables[i] = 1` nghĩa là `i` nằm trong miền giá trị, ngược lại sẽ không nằm trong miền giá trị ( `i` được xét từ 1 đến 9 ).

## 5: Xác định ô nào ưu tiên được điền trước

**Cách 1:** Dò lần lượt, ô nào trống thì chọn

```
Coord getNextEmptyCell(Sudoku sudoku) {
    int i, j;
    for (i = 0; i < NB_ROWS; i++) {
        for (j = 0; j < NB_COLUMNS; j++) {
            Coord pos = {i, j};
            if (sudoku.cells[i][j] == EMPTY) return pos;
        }
    }
}
```

**Cách 2:** Ưu tiên ô có miền giá trị ít nhất

```
Coord getNextMinDomainCell(Sudoku sudoku) {
    int minLength = INF;
    int i, j;
    Coord result;
    for (i = 0; i < NB_ROWS; i++) {
        for (j = 0; j < NB_COLUMNS; j++) {
            if (sudoku.cells[i][j] == EMPTY) {
                Coord pos = {i, j};
                int availablesLength = getAvailableValues(pos, sudoku).size;
                if (availablesLength < minLength) {
                    minLength = availablesLength;
                    result = pos;
                }
            }
        }
    }
    return result;
}
```

## 6: Giải quyết bài toán

Áp dụng thuật toán quay lui để giải quyết bài toán. Mỗi ô cần điền, điền lần lượt các số trong miền giá trị, ràng buộc các ô bị ảnh hưởng. Nếu không thể điền giá trị mới cho ô tiếp theo, ta thực hiện quay lui và thay đổi phương án khác. Cứ thế tiếp tục cho đến khi hoàn thành bài toán.

```
int exploredCounter = 0;
int sudokuBackTracking(Sudoku* sudoku) {
    if (isFilledSudoku(*sudoku)) return 1;
    // Coord position = getNextEmptyCell(*sudoku);
    Coord position = getNextMinDomainCell(*sudoku);
    List availables = getAvailableValues(position, *sudoku);
    if (availables.size == 0) {
        // Neu nhu mien gia tri rong nhung o van chua duoc dien
        return 0;
    }
    int j;
    for (j = 0; j < availables.size; j++) {
        int value = availables.data[j];
        sudoku->cells[position.x][position.y] = value;
        exploredCounter++;
        if (sudokuBackTracking(sudoku))
            return 1;
        sudoku->cells[position.x][position.y] = EMPTY;
    }
    return 0;
}
```

### 7: Lan truyền ràng buộc từ đề bài cho trước và kết thúc bài toán

Khi bắt đầu, ta lan truyền các ràng buộc từ các số cho trước, sau đó dùng giải thuật quay lui để hoàn thành bài toán.

```
Sudoku solveSudoku(Sudoku sudoku) {
    int i, j;
    clearConstrains(&sudoku.constrains);
    for (i = 0; i < NB_ROWS; i++) {
        for (j = 0; j < NB_COLUMNS; j++) {
            ListCoord history;
            initListCoord(&history);
            Coord pos = {i, j};
            spreadConstrainsFrom(pos, &sudoku.constrains, &history);
        }
    }
    exploredCounter = 0;
    if (sudokuBackTracking(&sudoku)) printf("Solved\n");
    else printf("Can not solve\n");
    printf("Explored %d states\n", exploredCounter);
    return sudoku;
}
```

#### Bảng Input đầu vào:

```
int inputs1[9][9] = {
    {5, 3, 0, 0, 7, 0, 0, 0, 0},
    {6, 0, 0, 1, 9, 5, 0, 0, 0},
    {0, 9, 8, 0, 0, 0, 0, 6, 0},
    {8, 0, 0, 0, 6, 0, 0, 0, 3},
    {4, 0, 0, 8, 0, 3, 0, 0, 1},
    {7, 0, 0, 0, 2, 0, 0, 0, 6},
    {0, 6, 0, 0, 0, 2, 8, 0, 0},
    {0, 0, 0, 4, 1, 9, 0, 0, 5},
    {0, 0, 0, 0, 8, 0, 0, 7, 9},
};
```

### 8. Hàm Main và đầu ra:

```
int main() {  
    Sudoku sudoku;  
    initSudokuWithValues(&sudoku, inputs1);  
    printSudoku(sudoku);  
    Sudoku result = solveSudoku(sudoku);  
    printSudoku(result);  
}
```

0 6 1	0 0 7	0 0 3
0 9 2	0 0 3	0 0 0
0 0 0	0 0 0	0 0 0
-----		
0 0 8	5 3 0	0 0 0
0 0 0	0 0 0	5 0 4
5 0 0	0 0 8	0 0 0
-----		
0 4 0	0 0 0	0 0 1
0 0 0	1 6 0	8 0 0
6 0 0	0 0 0	0 0 0
-----		
Solved		
Explored 110967 states		
Sudoku:		
-----		
4 6 1	9 8 7	2 5 3
7 9 2	4 5 3	1 6 8
3 8 5	2 1 6	4 7 9
-----		
1 2 8	5 3 4	7 9 6
9 3 6	7 2 1	5 8 4
5 7 4	6 9 8	3 1 2
-----		
8 4 9	3 7 5	6 2 1
2 5 3	1 6 9	8 4 7
6 1 7	8 4 2	9 3 5
-----		

## IV. Bài toán liên quan

### 1) Trò Chơi KenKen

Cho một bảng 4x4, trong đó có các khốii được tô đậm (gọi là các chuồng). Ở mỗi chuồng có một con số và một phép toán (nếu chuồng chỉ có 1 ô thì không có phép toán). Nhiệm vụ của người chơi là điền hết tất cả các ô trống thỏa mãn ràng buộc của bài toán.

2÷		7+	4
1-	3-		2-
		4×	
1-			

Các ràng buộc của trò chơi:

- Chỉ điền số từ 1 đến 4.
- Những số trên cùng hàng không được trùng nhau.
- Những số trên cùng cột không được trùng nhau.
- Trong chuồng, mỗi số được điền vào khi kết hợp với phép toán chuồng đó (không cần theo thứ tự) đúng bằng với con số được cho trước của chuồng đó.
- Nếu chuồng chỉ có đúng 1 ô thì ô đó có giá trị đúng bằng số gán cho chuồng đó.

### 2) Bài toán SEND + MORE = MONEY

Đây là một trò chơi suy luận toán học bao gồm các chữ cái và phép toán cho trước, mỗi chữ cái đại diện cho một chữ số từ 0 đến 9. Người chơi phải tìm ra được tất cả chữ số ẩn dưới chữ cái đó sao cho thỏa mãn các ràng buộc.

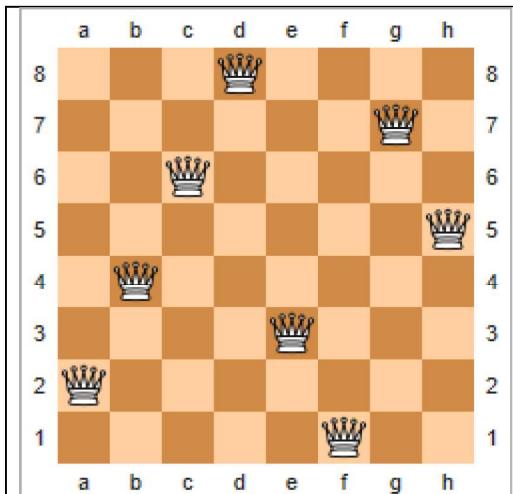
$$\begin{array}{r}
 & S & E & N & D \\
 + & M & O & R & E \\
 \hline
 M & O & N & E & Y
 \end{array}$$

Các ràng buộc của trò chơi:

- Mỗi chữ cái chỉ gồm số từ 0 đến 9.
- Mỗi chữ cái khác nhau đại diện cho một số duy nhất không trùng với chữ cái khác ( ví dụ giá trị của S phải khác giá trị của M).
- Chữ cái đầu tiên của mỗi số trong phép toán phải khác 0 ( S khác 0 và M khác 0)
- Tìm ra các số phải thỏa mãn với phép toán (cộng) từ đề bài.

### 3) Bài toán 8 quân hậu

Bài toán tám quân hậu là bài toán đặt tám quân hậu trên bàn cờ vua kích thước  $8 \times 8$  sao cho không có quân hậu nào có thể "ăn" được quân hậu khác. Như vậy, lời giải của bài toán là một cách xếp tám quân hậu trên bàn cờ sao cho không có hai quân nào đứng trên cùng hàng, hoặc cùng cột hoặc cùng đường chéo.



Các ràng buộc của trò chơi:

- Không có quân hậu nào đặt trên cùng một hàng.
- Không có quân hậu nào đặt trên cùng một cột.
- Không có quân hậu nào đặt trên cùng đường chéo.