Uva 00167

The Sultan's Successors

Time: 3 seconds

Problem Descriptions

- **The Sultan of Nubia has no children, so she has decided that the country will be split into up to *k* separate parts on her death and each part will be inherited by whoever performs best at some test.**
- **It is possible for any individual to inherit more than one or indeed all of the portions.**

Problem Descriptions

- To ensure that only highly intelligent people eventually become her successors, the Sultan has devised an ingenious test.
- In a large hall filled with the splash of fountains and the delicate scent of incense have been placed *k* chessboards.
- Each chessboard has numbers in the range 1 to 99 written on each square and is supplied with 8 jewelled chess queens.

Problem Descriptions

- The task facing each potential successor is to place the 8 queens on the chess board in such a way that no queen threatens another one, and so that the numbers on the squares thus selected sum to a number at least as high as one already chosen by the Sultan.
- (For those unfamiliar with the rules of chess, this implies that each row and column of the board contains exactly one queen, and each diagonal contains no more than one.)

Problem Descriptions

- **Write a program that will read in the number and details of the chessboards and determine the highest scores possible for each board under these conditions.**
- **(You know that the Sultan is both a good chess player and a good mathematician and you suspect that her score is the best attainable.)**

Input

- **Input will consist of *k* (the number of boards), on a line by itself, followed by *k* sets of 64 numbers, each set consisting of eight lines of eight numbers.**
- **Each number will be a positive integer less than 100.**
- **There will never be more than 20 boards.**

Output (1/2)

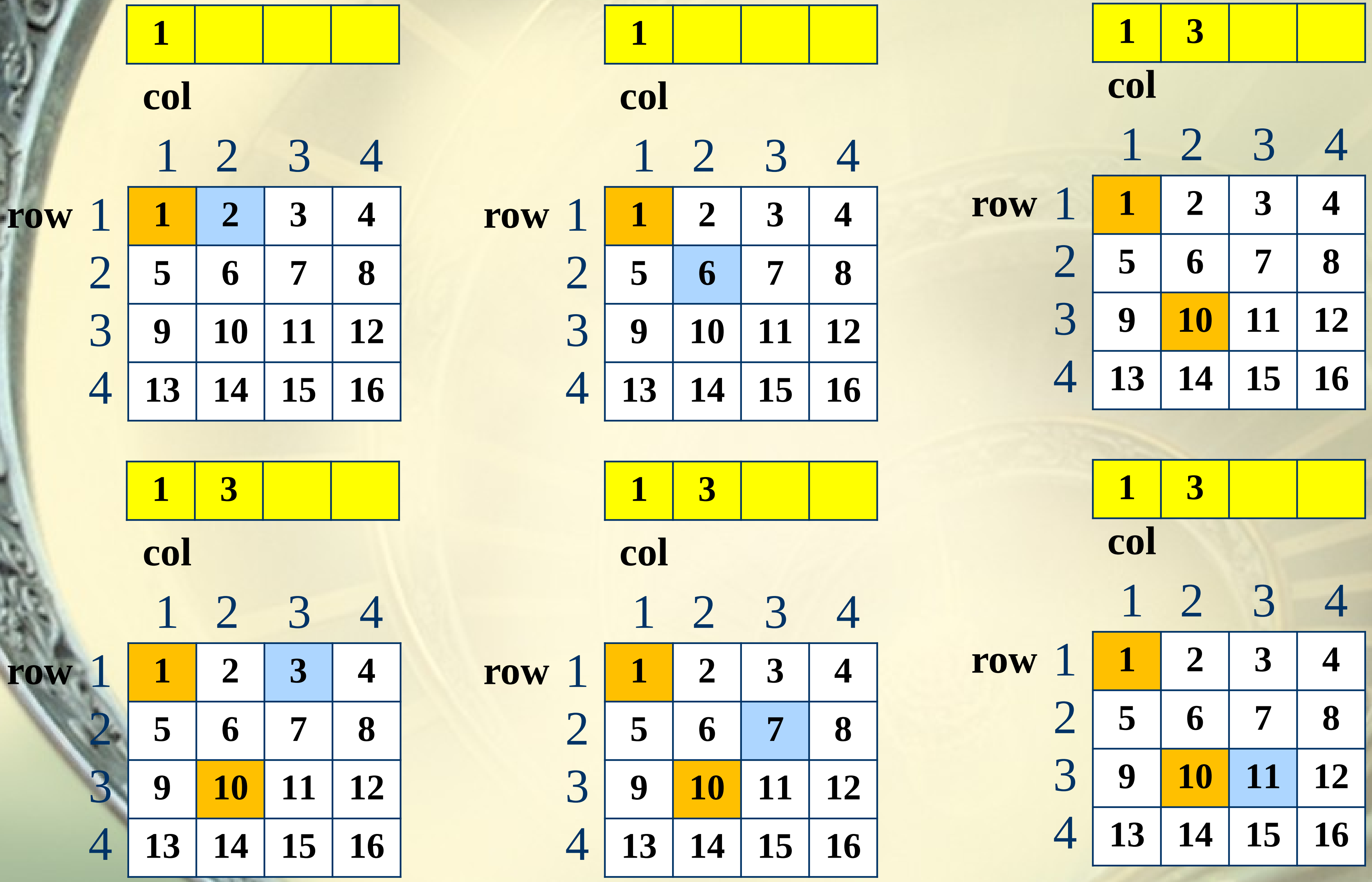
- **Output will consist of *k* numbers consisting of your *k* scores, each score on a line by itself and right justified in a field 5 characters wide.**

Sample Input / Output

```
1
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

```
3 3 3 3 4
```

Backtracking and Recursive



1			

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1			

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	3		

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	3		

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	3		

col

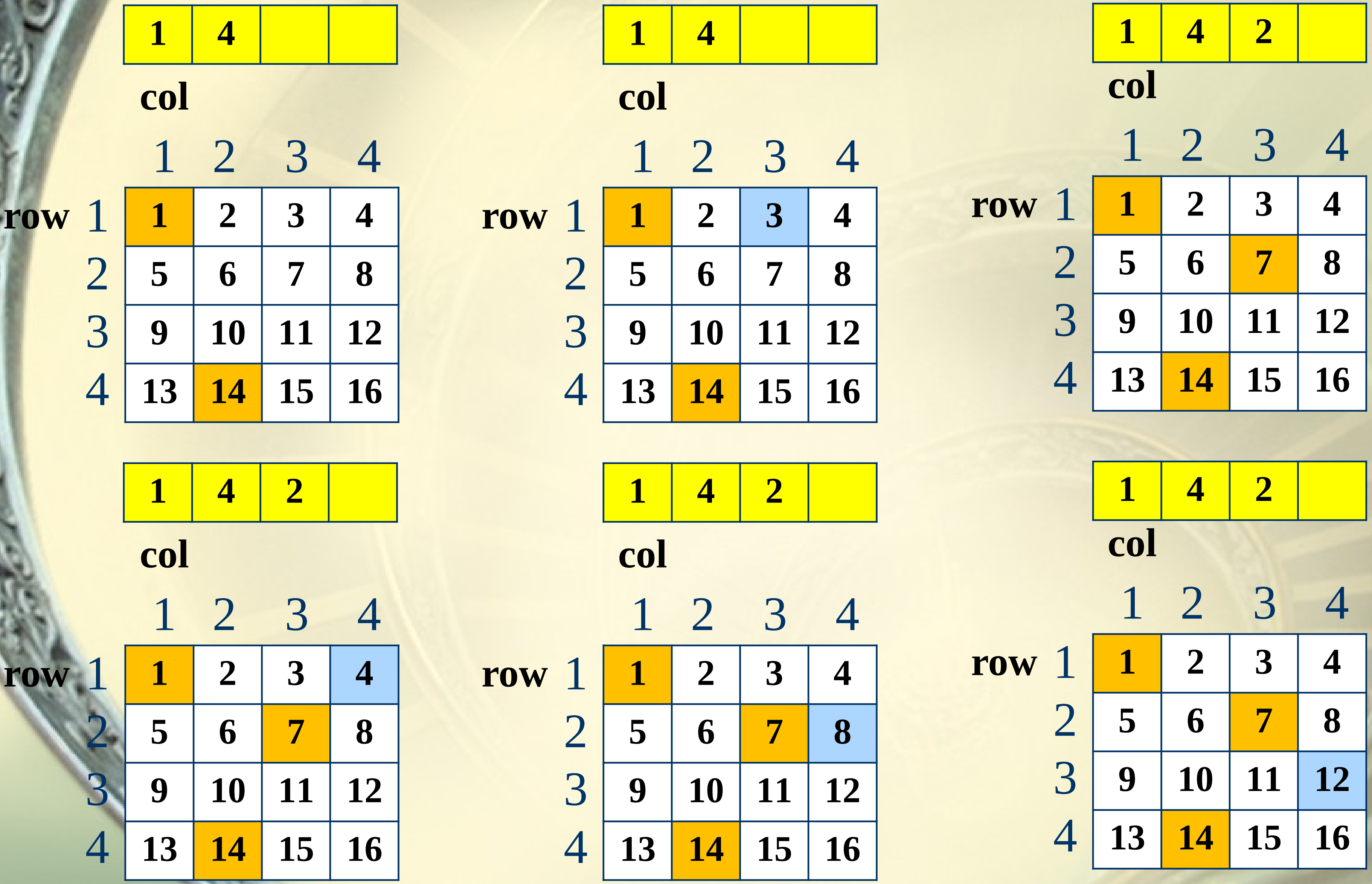
	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	3		

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

Backtracking and Recursive



1	4		

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	4		

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	4	2	

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	4	2	

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	4	2	

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

1	4	2	

col

	1	2	3	4
row 1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

Backtracking and Recursive

1	4	2	

col

row	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

2	4		

col

row	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

5+14+3+12=34

2	4	1	3

col

row	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

3			

col

row	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

3	1		

col

row	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

9+2+15+8=34

3	1	4	2

col

row	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

Backtracking and Recursive

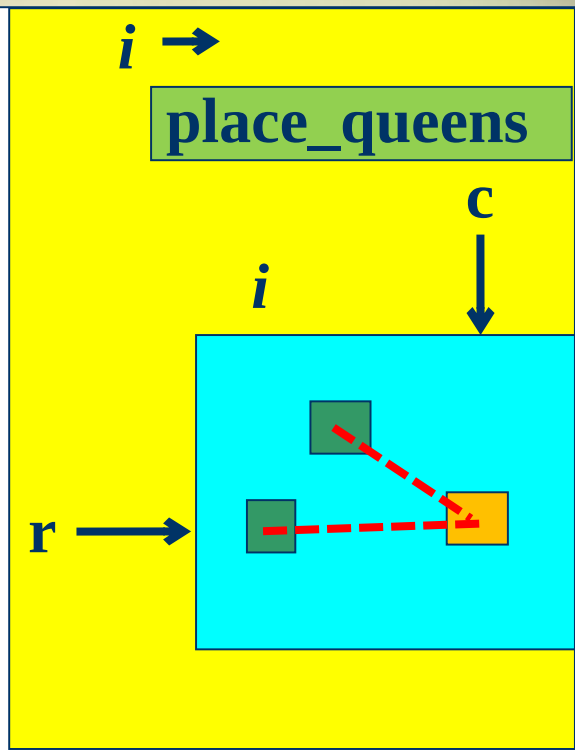
```
41  int main()
42  {
43      scanf("%d", &k);
44      while (k--)
45      {   for(int i = 1; i <= 8; i++)
46          {
47              for (int j = 1; j <= 8; j++)
48              scanf("%d", &chessboard[i][j]);
49          }
50          tmax = 0;
51          dfs(1);
52          printf("%5d\n", tmax);
53      }
54  }
```

Backtracking and Recursive

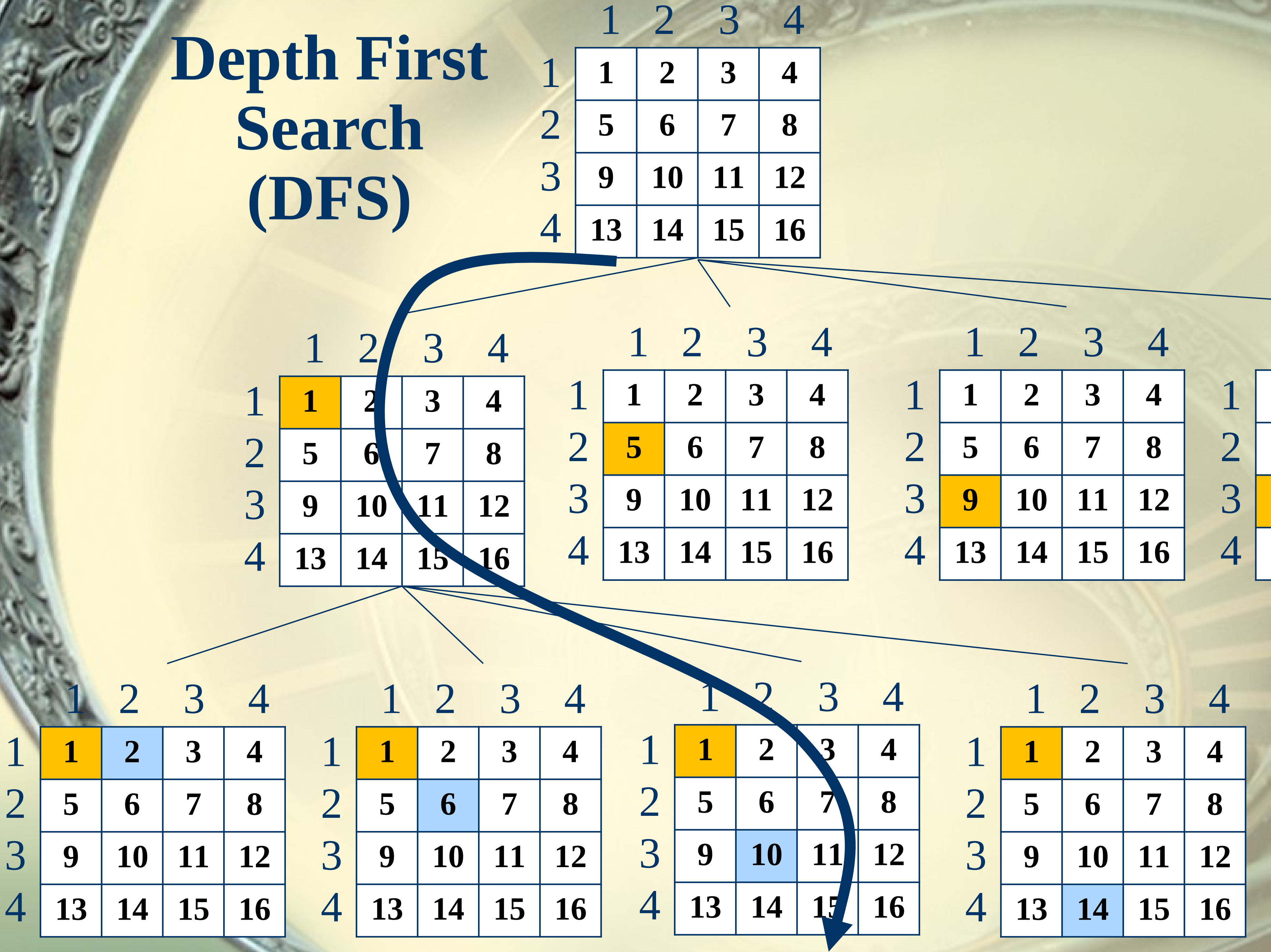
```
21  void dfs(int c)
22  {
23      if(c == 8)
24      {
25          int tot = 0;
26          for(int i = 1; i <= 8; i++)
27              tot += chessboard[i][place_queens[i]];
28          tmax = max(tot, tmax);
29          return;
30      }
31      for(int r = 1; r <= 8; r++)
32      {
33          if(!conflict(c, r))
34          {
35              place_queens[c] = r;
36              dfs(c + 1);
37          }
38      }
39  }
```

place_queens[i]

```
#include <iostream>

using namespace std;

int  k, tmax;
int  chessboard[10][10];
int  place_queens[10];

bool conflict (int c, int r)
{
    for(int i = 1; i <= c; i++)
    {
        if(i==c || place_queens[i] == r)
            return true;
        if(abs(c - i) == abs(place_queens[i] - r))
            return true;
    }
    return false;
}
```



Depth First Search (DFS)



	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16