回溯與剪枝

日月卦長

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

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

- 每一列的數字均須包含 1～9，不能缺少，也不能重複。
- 每一宮(粗黑線圍起來的區域，通常是 3*3 的九宮格)的數字均須包含 1～9，不能缺少，也不能重複。

Sudoku 數獨

判斷數獨是否合法

Input

```
193265478
782314956
456978132
234851697
965437281
871692345
319586724
527143869
648729513
```

Output

```
Yes
```

判斷數獨是否合法

Input

```
123456789
234567891
345678912
456789123
567891234
678912345
789123456
891234567
912345678
```

Output

```
No
```

輸入資料

```cpp
#include <iostream>
#include <string>
using namespace std;

int grid[9][9];
void input() {
  for (int r = 0; r < 9; ++r) {
    string buffer;
    cin >> buffer;
    for (int c = 0; c < 9; ++c) {
      grid[r][c] = buffer[c] - '0';
    }
  }
}
```

判斷 row

3	2	3	9	5	7				

```
bool row[9][10]
```

row[3]

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

判斷 column

7

							1	
							3	
							5	
							7	
							9	

```
bool col[9][10]
```

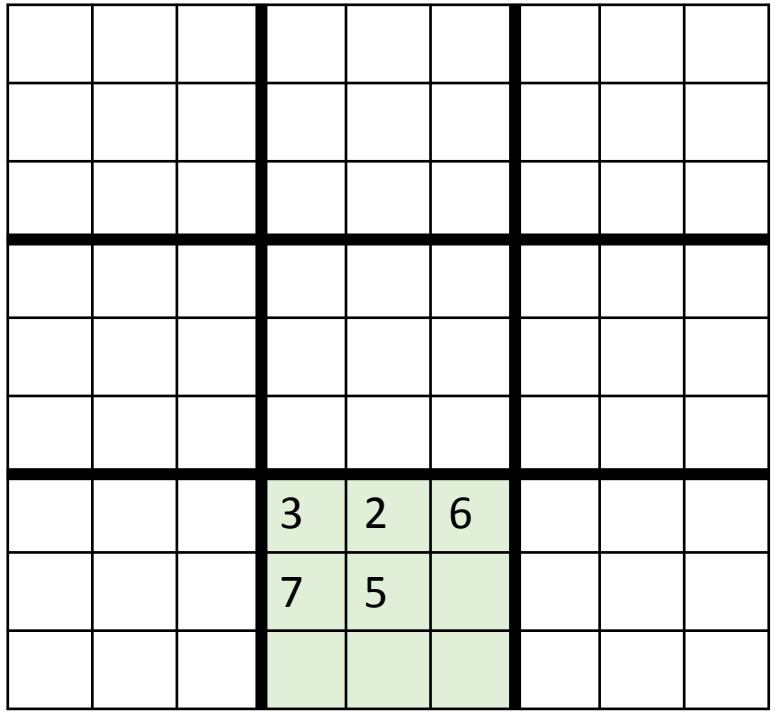
col[7]

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

判斷 subgrid

1

2



subgrids[2][1]

```
bool subgrids[3][3][10];
```

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

判斷 + 更新資料

```
bool row[9][10], col[9][10];
bool subgrids[3][3][10];

bool illegal(int r, int c, int num) {
  return row[r][num] || col[c][num] || subgrids[r / 3][c / 3][num];
}

void update(int r, int c, int num, bool val) {
  row[r][num] = val;
  col[c][num] = val;
  subgrids[r / 3][c / 3][num] = val;
}
```

判斷數獨

```cpp
bool check() {
  for (int r = 0; r < 9; ++r) {
    for (int c = 0; c < 9; ++c) {
      if (grid[r][c] == 0) continue; // 伏筆
      if (illegal(r, c, grid[r][c]))
        return false;
      update(r, c, grid[r][c], true);
    }
  }
  return true;
}

int main() {
  input();
  cout << (check() ? "Yes\n" : "No\n");
  return 0;
}
```

數獨求解(輸出最小字典序)

Input

```
1.....478
7..314956
456978132
234851697
965437281
871692345
319586724
527143869
648729513
```

Output

```
193265478
782314956
456978132
234851697
965437281
871692345
319586724
527143869
648729513
```

輸入資料

```
int grid[9][9];
void input() {
  for (int r = 0; r < 9; ++r) {
    string buffer;
    cin >> buffer;
    for (int c = 0; c < 9; ++c) {
      if (isdigit(buffer[c]))
        grid[r][c] = buffer[c] - '0';
      else
        grid[r][c] = 0;
    }
  }
}
```

印出答案

```
void print() {
  for (int r = 0; r < 9; ++r) {
    for (int c = 0; c < 9; ++c)
      cout << grid[r][c];
    cout << '\n';
  }
}
```

幫每個格子編號

- $R = 6$
- $C = 3$

$$57 = R \times 9 + C$$
$$= 6 \times 9 + 3$$

- $R = [57/9] = 6$
- $C = 57\%9 = 3$

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

暴力枚舉所有可能性

```
bool dfs(int idx) {
  if (idx == 81) {
    memset(row, 0, sizeof(row));
    memset(col, 0, sizeof(col));
    memset(subgrids, 0, sizeof(subgrids));
    return check();
  }
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    grid[r][c] = num;
    if (dfs(idx + 1)) return true;
  }
  grid[r][c] = 0;
  return false;
}
```

暴力枚舉所有可能性

- 假設 $0 \sim idx-1$ 的格子都填好了
- 依序枚舉所有填滿 $idx \sim 80$ 的所有可能
- 若找到合法解則回傳 true

```
bool dfs(int idx) {
  if (idx == 81) {
    memset(row, 0, sizeof(row));
    memset(col, 0, sizeof(col));
    memset(subgrids, 0, sizeof(subgrids));
    return check();
  }
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    grid[r][c] = num;
    if (dfs(idx + 1)) return true;
  }
  grid[r][c] = 0;
  return false;
}
```

暴力枚舉所有可能性

- 所有格子都填滿了
- 回傳是否是合法數獨

```
bool dfs(int idx) {
  if (idx == 81) {
    memset(row, 0, sizeof(row));
    memset(col, 0, sizeof(col));
    memset(subgrids, 0, sizeof(subgrids));
    return check();
  }
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    grid[r][c] = num;
    if (dfs(idx + 1)) return true;
  }
  grid[r][c] = 0;
  return false;
}
```

暴力枚舉所有可能性

- 如果格子已經有數字了
- 就直接跳過枚舉下一個格子

```
bool dfs(int idx) {
  if (idx == 81) {
    memset(row, 0, sizeof(row));
    memset(col, 0, sizeof(col));
    memset(subgrids, 0, sizeof(subgrids));
    return check();
  }
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    grid[r][c] = num;
    if (dfs(idx + 1)) return true;
  }
  grid[r][c] = 0;
  return false;
}
```

暴力枚舉所有可能性

- 枚舉數字 1~9
  依序填入格子中

- 由於由小到大枚舉
  一旦找到解那就會是
  字典序最小的解

```
bool dfs(int idx) {
  if (idx == 81) {
    memset(row, 0, sizeof(row));
    memset(col, 0, sizeof(col));
    memset(subgrids, 0, sizeof(subgrids));
    return check();
  }
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    grid[r][c] = num;
    if (dfs(idx + 1)) return true;
  }
  grid[r][c] = 0;
  return false;
}
```

暴力枚舉所有可能性

- 為了讓當前的函數不影響其他正在遞迴的函數
- 結束遞迴時一定要把所有修改都復原

```
bool dfs(int idx) {
  if (idx == 81) {
    memset(row, 0, sizeof(row));
    memset(col, 0, sizeof(col));
    memset(subgrids, 0, sizeof(subgrids));
    return check();
  }
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    grid[r][c] = num;
    if (dfs(idx + 1)) return true;
  }
  grid[r][c] = 0;
  return false;
}
```

暴力枚舉所有可能性

```
int main() {
  input();
  if (check() && dfs(0))
    print();
  else
    cout << "No answer\n";
  return 0;
}
```

更難的測資

Input

```
.1.....9
...3..8..
......6..
....124..
7.3......
5........
8..6.....
....4..2.
...7...5.
```

Output

```
318456279
267391845
459287613
986512437
723964581
541873962
872635194
635149728
194728356
```

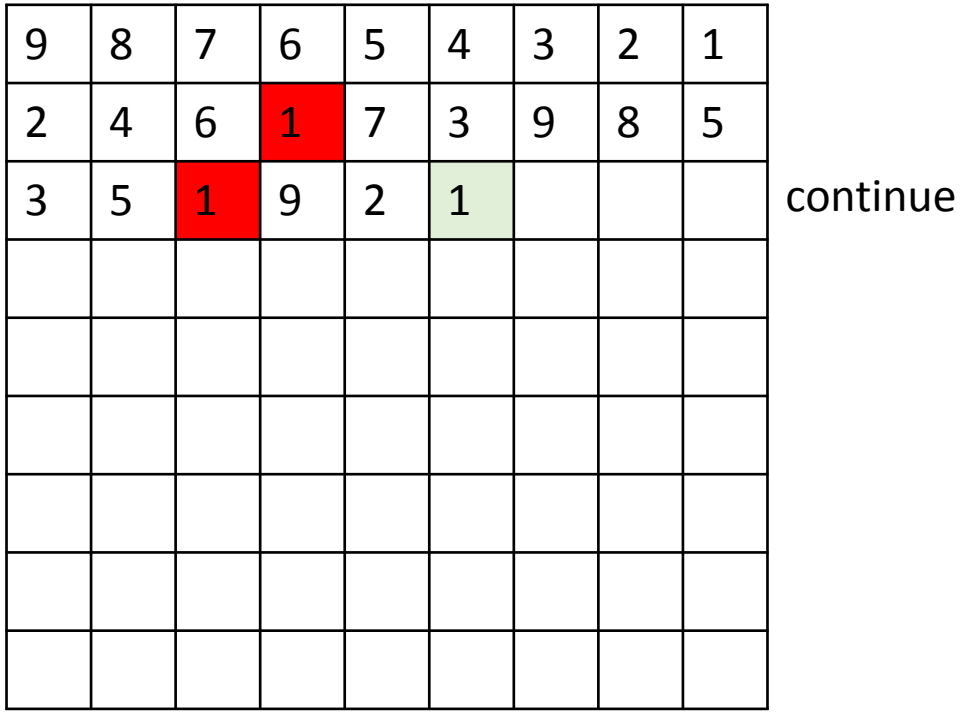
回朔 = 暴力枚舉 + 剪枝 (Backtracking)

進入遞迴前就發現走下去永遠找不到解
就直接跳過這次遞迴

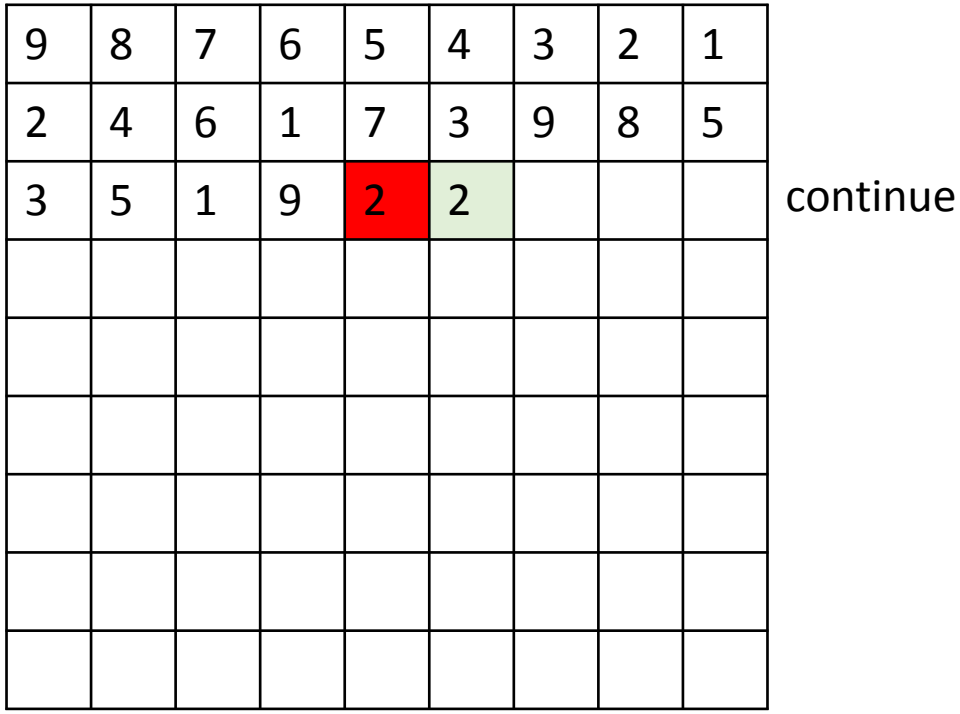
剪枝: 不要做沒意義的枚舉

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

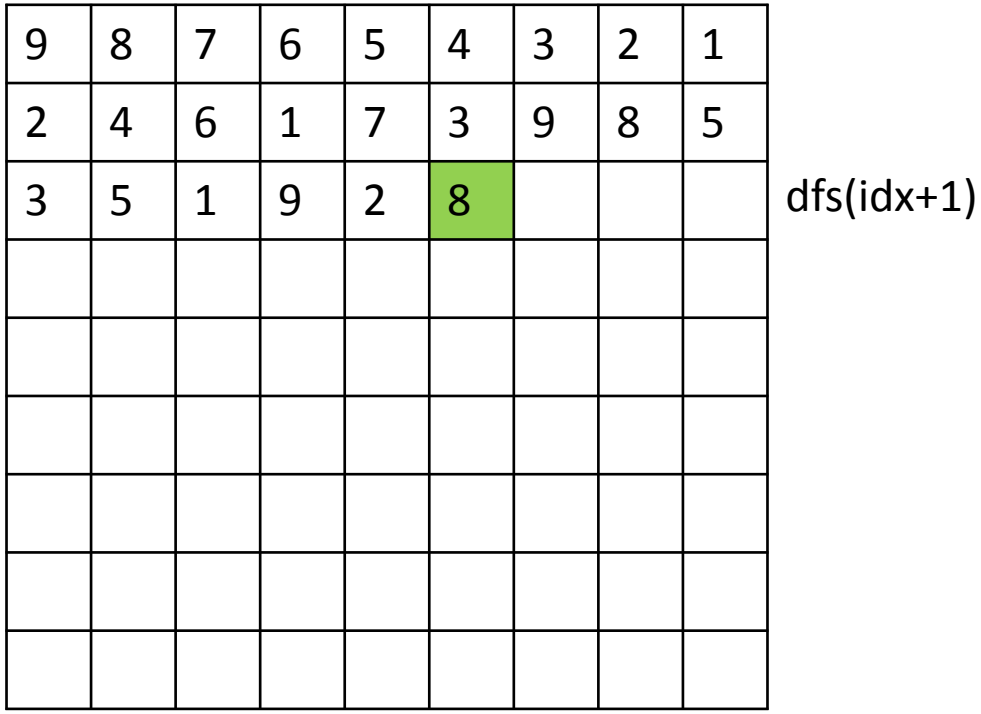
剪枝: 不要做沒意義的枚舉



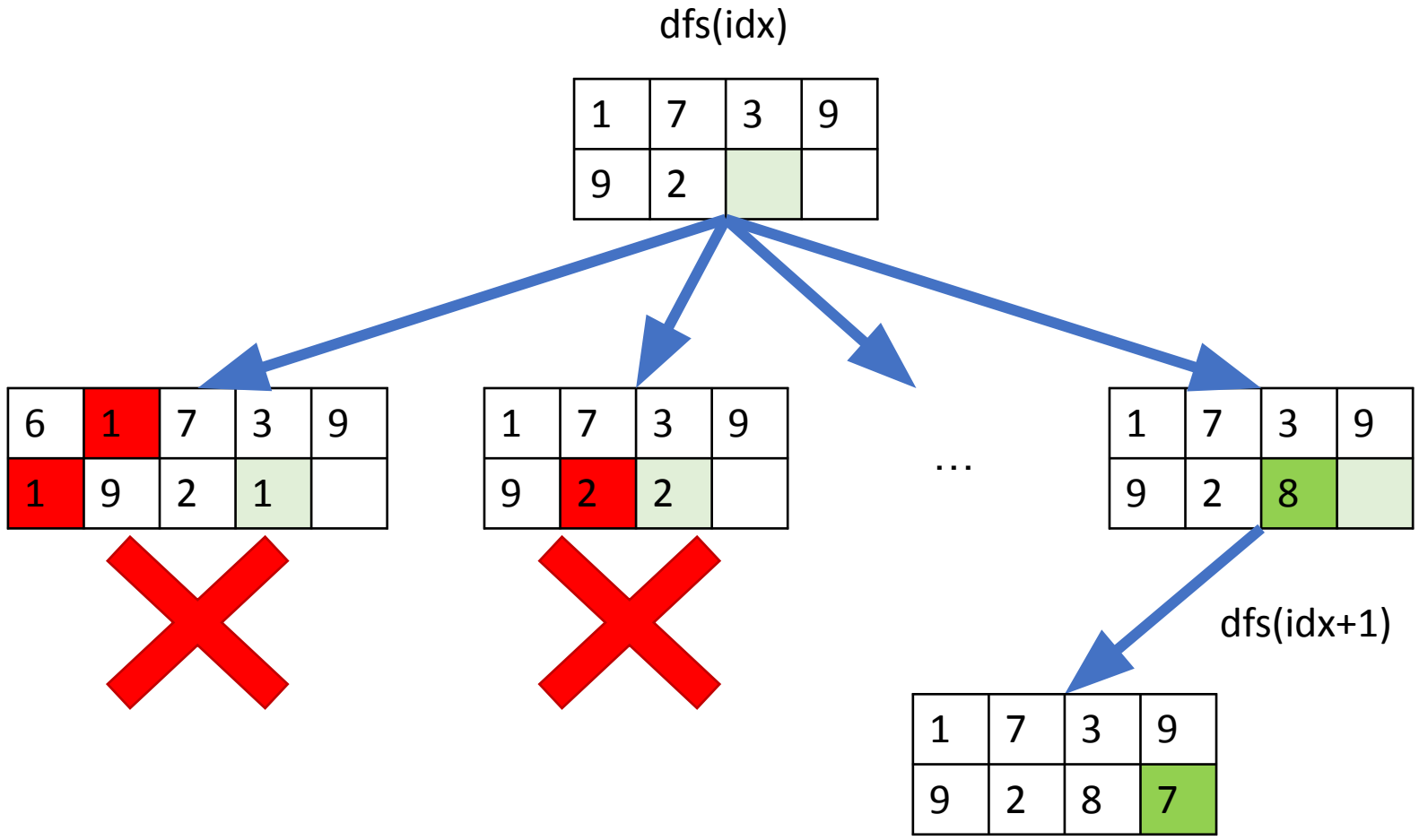
剪枝: 不要做沒意義的枚舉



剪枝: 不要做沒意義的枚舉



剪枝: 不要做沒意義的枚舉



使用 Backtracking

- 如果 grid[r][c] = num 時
  就可以直接判斷是非法解

- 就沒必要針對 num 遞迴
  直接跳過 (continue)

```
bool dfs(int idx) {
  if (idx == 81) return true;
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    if (illegal(r, c, num)) continue;
    grid[r][c] = num;
    update(r, c, num, true);
    if (dfs(idx + 1)) return true;
    update(r, c, num, false);
  }
  grid[r][c] = 0;
  return false;
}
```

使用 Backtracking

- 進入遞迴前紀錄當前格子填入 num
- 若沒找到答案
  離開遞迴後把紀錄刪除

```
bool dfs(int idx) {
  if (idx == 81) return true;
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    if (illegal(r, c, num)) continue;
    grid[r][c] = num;
    update(r, c, num, true);
    if (dfs(idx + 1)) return true;
    update(r, c, num, false);
  }
  grid[r][c] = 0;
  return false;
}
```

使用 Backtracking

- 由於每次遞迴前都有判斷合法性
- 當 81 個格子都填完後答案一定是合法的
- 直接 return true

```
bool dfs(int idx) {
  if (idx == 81) return true;
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = 1; num <= 9; ++num) {
    if (illegal(r, c, num)) continue;
    grid[r][c] = num;
    update(r, c, num, true);
    if (dfs(idx + 1)) return true;
    update(r, c, num, false);
  }
  grid[r][c] = 0;
  return false;
}
```

再難一點?

Input

```
.........
.....3.85
..1.2....
...5.7...
..4...1..
.9.......
5......73
..2.1....
....4...9
```

Output

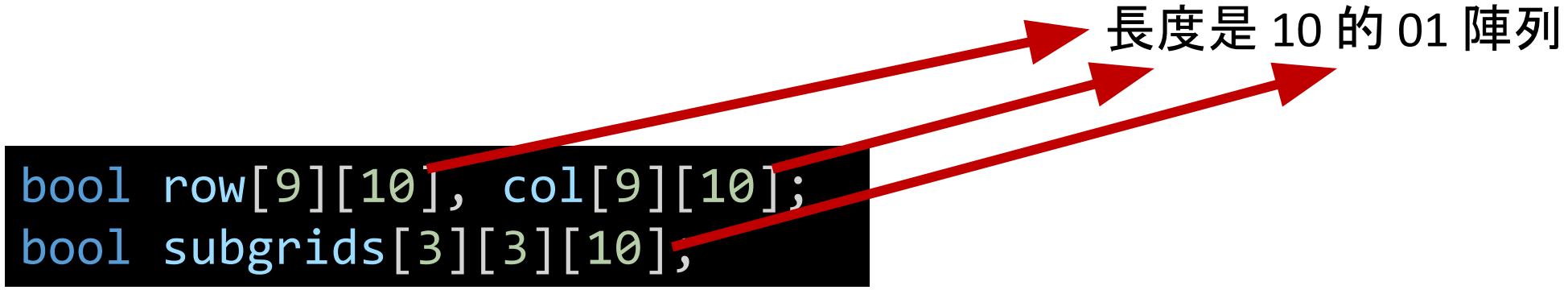
```
987654321
246173985
351928746
128537694
634892157
795461832
519286473
472319568
863745219
```

就是 0 和 1

true □ 1

false □ 0

為什麼不用二進位存?




```
bool row[9][10], col[9][10];
bool subgrids[3][3][10];
```

```
int row[9], col[9];
int subgrids[3][3];
```

int: 一般電腦上是 32 個 bit 組成
□長度是 32 的 01 陣列

二進位表示法

	二進位	十進位
1	0000000010	2
2	0000000100	4
3	0000001000	8
4	0000010000	16
5	0000100000	32
6	0001000000	64
7	0010000000	128
8	0100000000	256
9	1000000000	512

```
int lg(int x) {
  switch(x){
    case 2: return 1;
    case 4: return 2;
    case 8: return 3;
    case 16: return 4;
    case 32: return 5;
    case 64: return 6;
    case 128: return 7;
    case 256: return 8;
    case 512: return 9;
  }
  return -1;
}
```

```
cout << lg(1 << 8) << endl;
cout << __lg(1 << 8) << endl;
```

黑魔法

輸入資料

```
int grid[9][9];
void input() {
  for (int r = 0; r < 9; ++r) {
    string buffer;
    cin >> buffer;
    for (int c = 0; c < 9; ++c) {
      if (isdigit(buffer[c]))
        grid[r][c] = 1 << (buffer[c] - '0');
      else
        grid[r][c] = 0;
    }
  }
}
```

輸出答案

```
void print() {
  for (int r = 0; r < 9; ++r) {
    for (int c = 0; c < 9; ++c)
      cout << __lg(grid[r][c]);
    cout << '\n';
  }
}
```

判斷 + 更新資料

```
int row[9], col[9];
int subgrids[3][3];

bool illegal(int r, int c, int num) {
  return (row[r] | col[c] | subgrids[r / 3][c / 3]) & num;
}

void update(int r, int c, int num) {
  row[r] ^= num;
  col[c] ^= num;
  subgrids[r / 3][c / 3] ^= num;
}
```

透過二進位紀錄用過的數字

- 整體上沒太大差別
- 記得枚舉數字時要用二進位

```
bool dfs(int idx) {
  if (idx == 81) return true;
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  for (int num = (1 << 1); num <= (1 << 9); num <<= 1) {
    if (illegal(r, c, num)) continue;
    grid[r][c] = num;
    update(r, c, num);
    if (dfs(idx + 1)) return true;
    update(r, c, num);
  }
  grid[r][c] = 0;
  return false;
}
```

使用 lowbit 減少枚舉數量

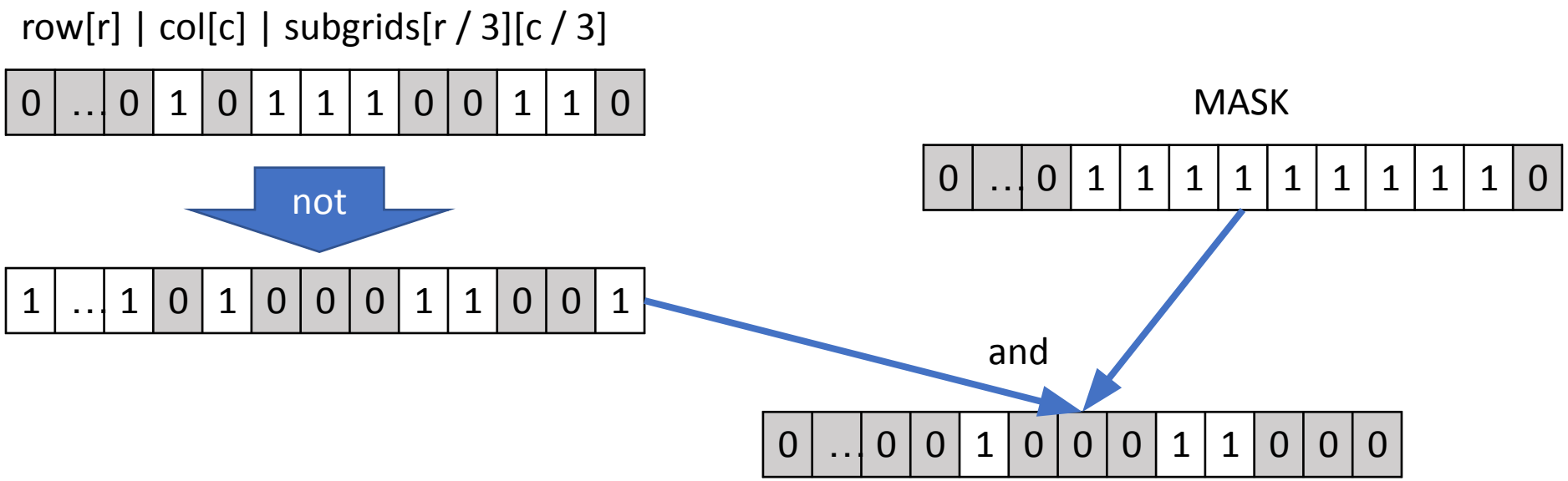
lowbit 優化

不能用的數字集合

- 紅色區域計算後的數字
  若第 k 個 bit 是 1，表示數字 k 不能被使用

```
bool illegal(int r, int c, int num) {
  return (row[r] | col[c] | subgrids[r / 3][c / 3]) & num;
}
```

可以用的數字集合




```
const int MASK = (1 << 10) - 2;
int S = MASK & ~(row[r] | col[c] | subgrids[r / 3][c /
3]);
```

可以用的數字集合

```
const int MASK = (1 << 10) - 2;
bool dfs(int idx) {
  if (idx == 81) return true;
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  int S = MASK & ~(row[r] | col[c] | subgrids[r / 3][c / 3]);
  for (int num = (1 << 1); num <= (1 << 9); num <<= 1) {
    if ((num & S) == 0) continue;
    grid[r][c] = num;
    update(r, c, num);
    if (dfs(idx + 1)) return true;
    update(r, c, num);
  }
  grid[r][c] = 0;
  return false;
}
```

重要函數 $lowbit(x)$

- $lowbit(x)$:
  非負整數 x 在二進位表示時，最靠右邊的 1 所對應的值。

- 範例：

$$20_{(10)} = 10100_{(2)}$$

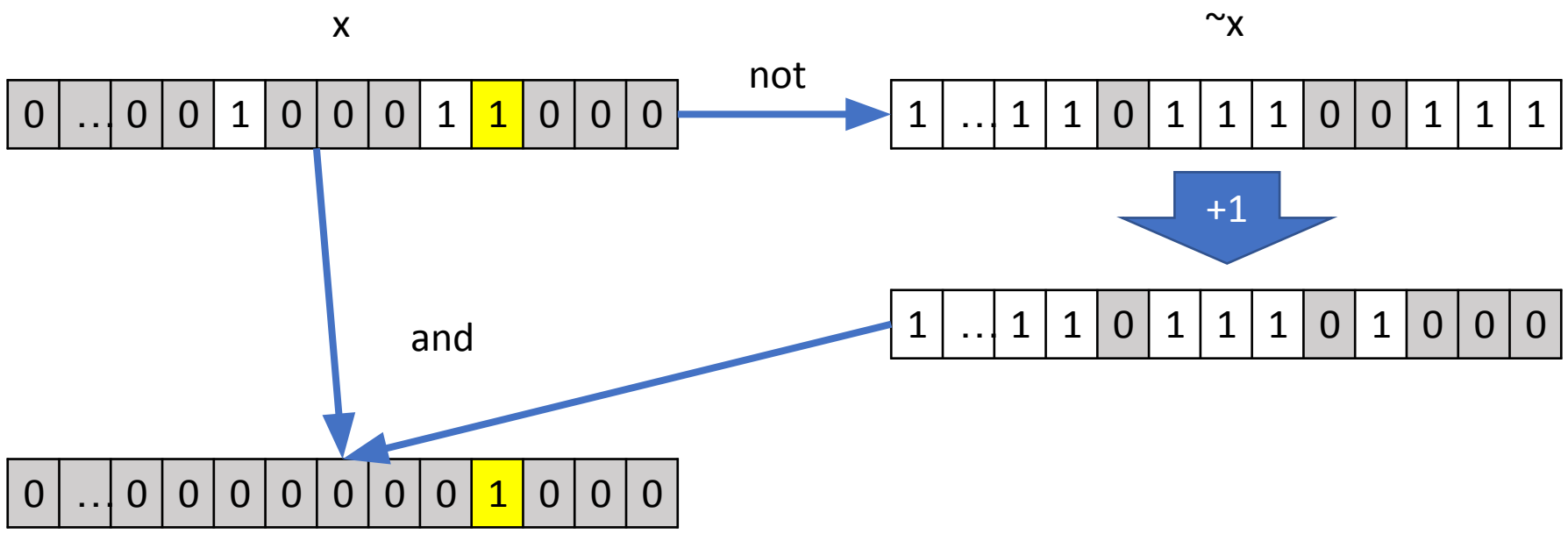
  其中的兩個 bit 分別表示 2^4 和 2^2，因此

$$lowbit(20) = 2^2 = 4$$

$lowbit(x)$ 計算

二補數的 -x

```
int lowbit(int x) { return x & (~x + 1); }
```




```
int lowbit(int x) { return x & -x; }
```

Unspecific Behavior (before C++20)

枚舉所有是 1 的 bit

計算 32 次

```cpp
#include <bitset>
#include <iostream>
using namespace std;

int main() {
  int S = 0b100011000;
  cout << bitset<32>(S) << endl;
  for (int i = 0; i < 32; ++i) {
    if (S & (1 << i))
      cout << bitset<32>(1 << i) << endl;
  }
  return 0;
}
```

計算 3(是 1 的 bit 數) 次

```cpp
#include <bitset>
#include <iostream>
using namespace std;
int lowbit(int x) { return x & -x; }
int main() {
  int S = 0b100011000;
  cout << bitset<32>(S) << endl;
  for (int num = 0; S; S ^= num) {
    num = lowbit(S);
    cout << bitset<32>(num) << endl;
  }
  return 0;
}
```

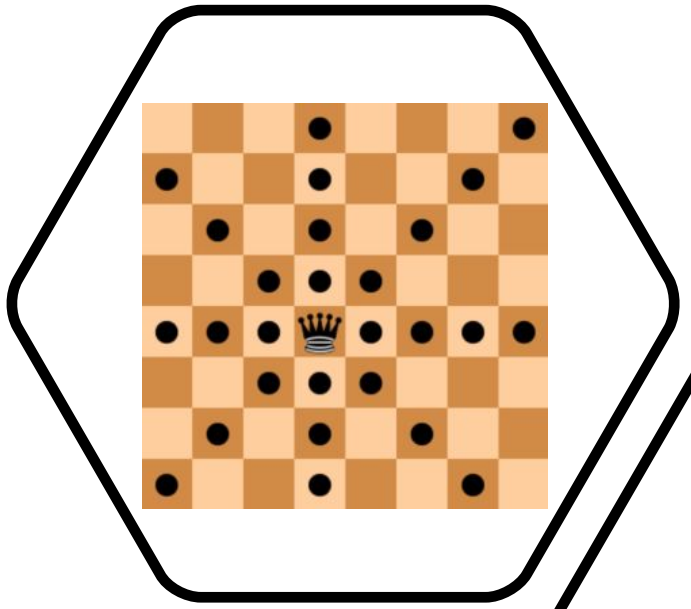
lowbit 優化

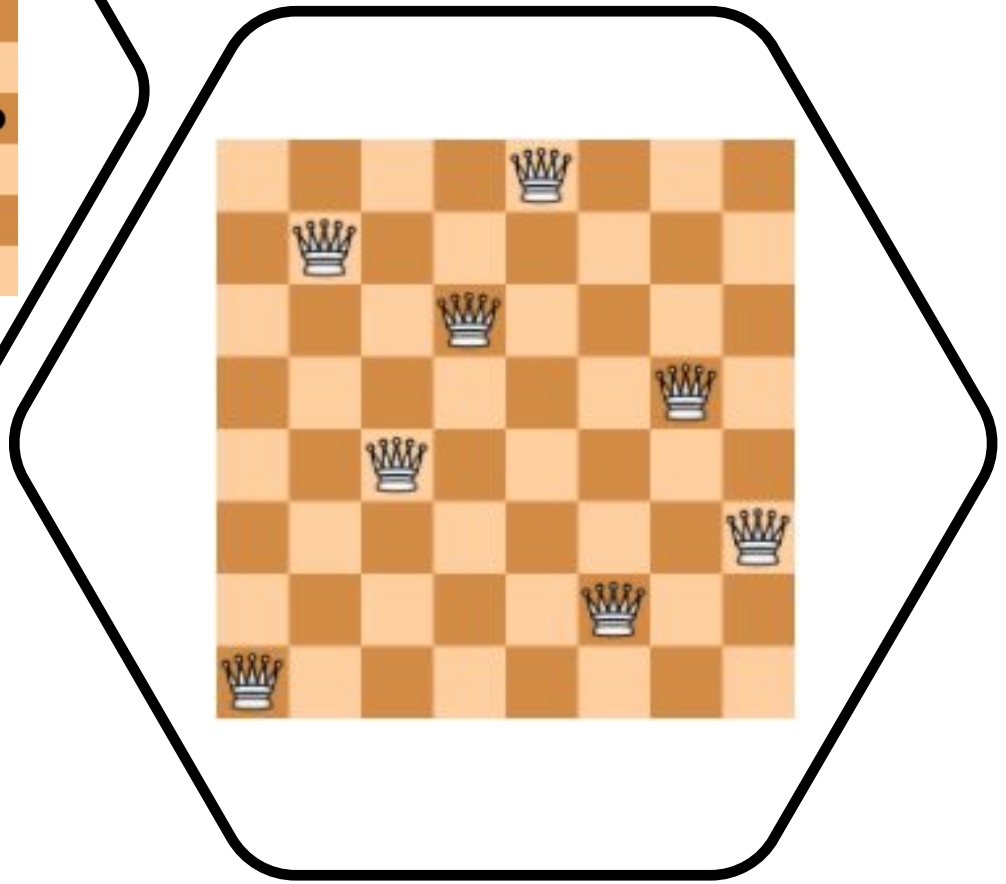
```
const int MASK = (1 << 10) - 2;
int lowbit(int x) { return x & -x; }
bool dfs(int idx) {
  if (idx == 81) return true;
  int r = idx / 9, c = idx % 9;
  if (grid[r][c]) return dfs(idx + 1);
  int S = MASK & ~(row[r] | col[c] | subgrids[r / 3][c / 3]);
  for (int num = 0; S; S ^= num) {
    num = lowbit(S);
    grid[r][c] = num;
    update(r, c, num);
    if (dfs(idx + 1)) return true;
    update(r, c, num);
  }
  grid[r][c] = 0;
  return false;
}
```

更難的數獨

- https://www.spoj.com/problems/SUDOKU/

- X 演算法
  - Dancing Links





n皇后問題

- 在n x n的棋盤上，擺上n個皇后
- 皇后能「吃掉」的範圍是米字形，如上圖
- 問你這些皇后有幾種擺法使得他們不會互相「吃掉」
- 右圖是8皇后的其中一組解

輸出 n 皇后的所有可能數

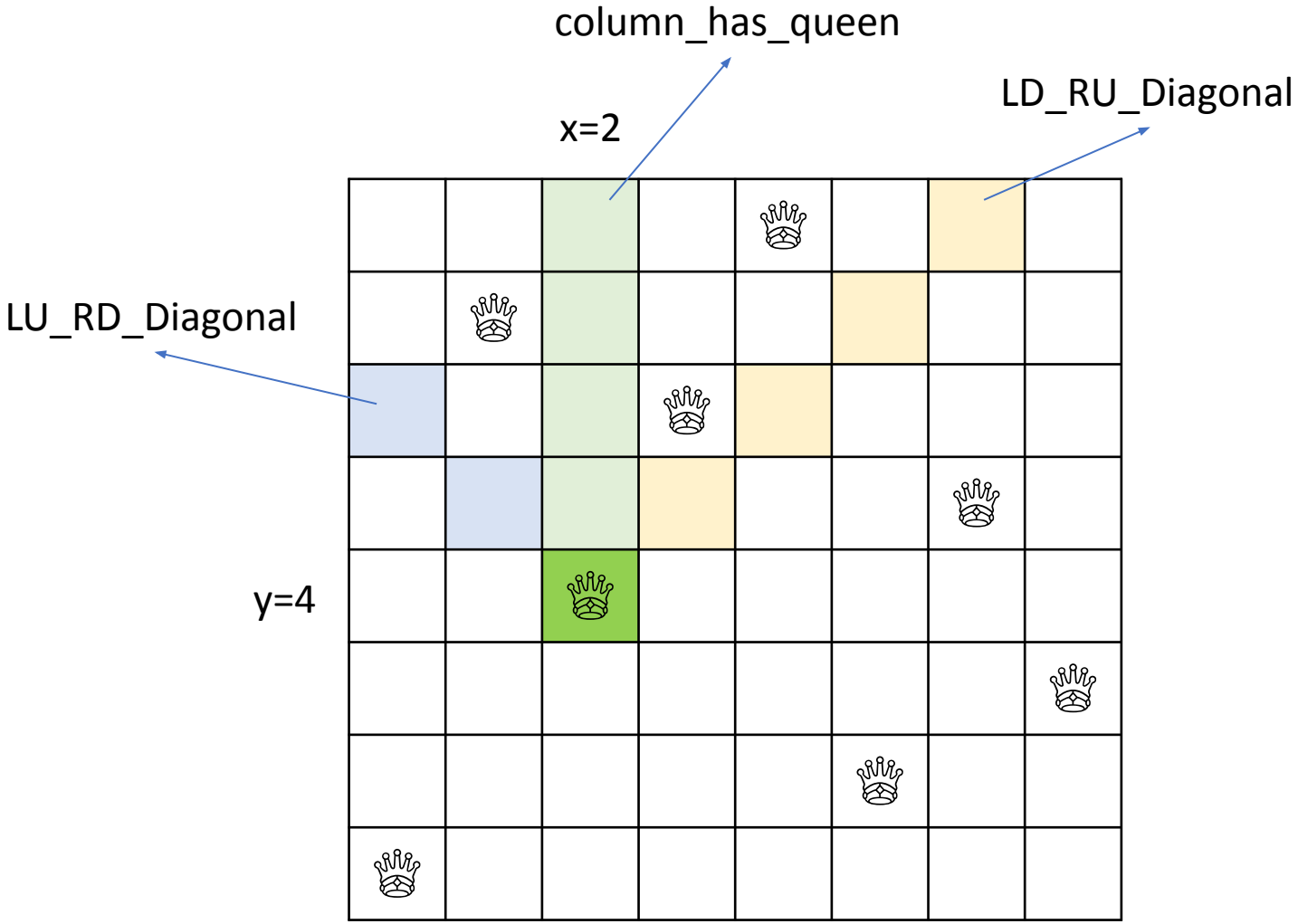
Input

8

Output

92

需要判斷的東西



需要判斷的東西

```cpp
#include <iostream>
using namespace std;

const int MAXN = 20;

bool column_has_queen[MAXN];
bool LD_RU_Diagonal[MAXN * 2 + 1];
bool LU_RD_Diagonal[MAXN * 2 + 1];
```

對角線的表?

```
int n = 5;
auto LD_RU_Diagonal = [&](int y, int x)
  { return (y + x); };
auto LU_RD_Diagonal = [&](int y, int x)
  { return n - 1 + (y - x); };
void show_table(auto callback) {
  for (int y = 0; y < n; ++y) {
    for (int x = 0; x < n; ++x)
      cout << callback(y, x) << ' ';
    cout << '\n';
  }
  cout << '\n';
}
show_table(LD_RU_Diagonal);
show_table(LU_RD_Diagonal);
```

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

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

判斷 + 更新資料

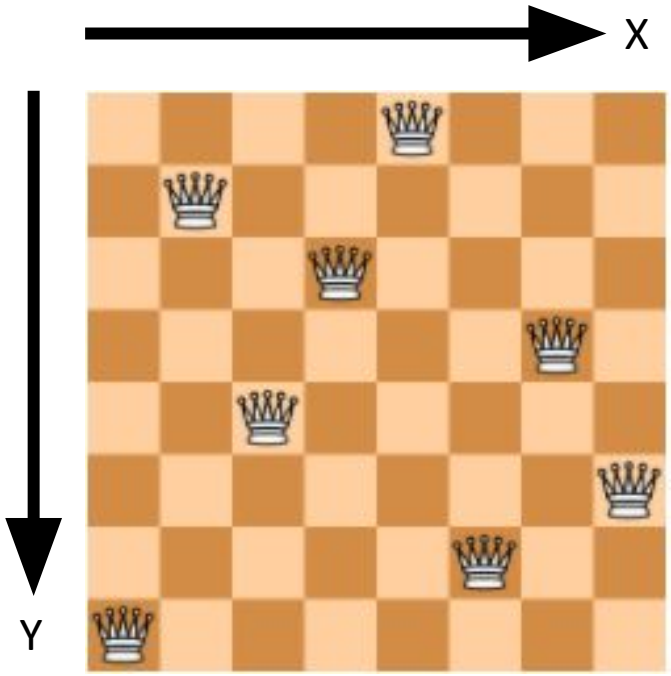
```
int n; // input

void update(int y, int x, bool val) {
  column_has_queen[x] = val;
  LD_RU_Diagonal[y + x] = val;
  LU_RD_Diagonal[n - 1 + (y - x)] = val;
}

bool isValidQueenPosition(int y, int x) {
  if (column_has_queen[x])
    return false;
  if (LD_RU_Diagonal[y + x])
    return false;
  if (LU_RD_Diagonal[n - 1 + (y - x)])
    return false;
  return true;
}
```

遞迴找出所有答案

```
int ans;
void dfs(int y) {
  if (y == n) {
    ++ans;
    return;
  }
  for (int x = 0; x < n; ++x) {
    if (!isValidQueenPosition(y, x))
      continue;
    update(y, x, true);
    dfs(y + 1);
    update(y, x, false);
  }
}
```

```
int main() {
  cin >> n;
  dfs(0);
  cout << ans <<
endl;
  return 0;
}
```

解的表示法



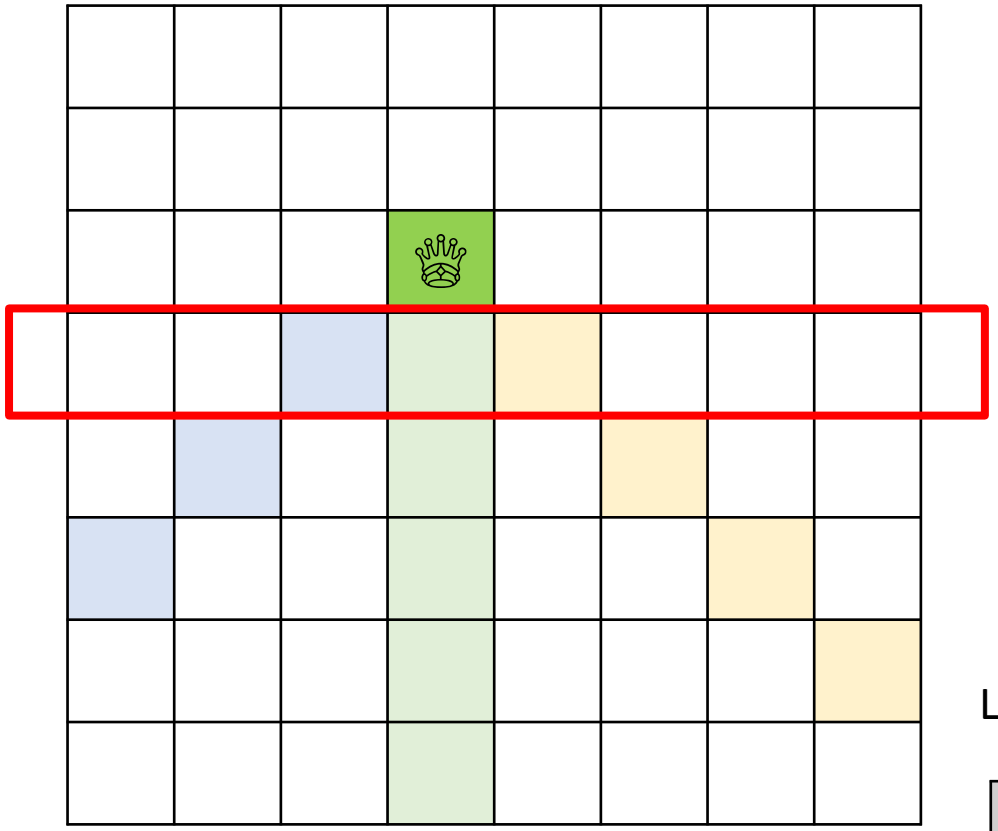
n 皇后的解會是某個 $0 \sim n-1$ 的全排列

row = [4, 1, 3, 6, 2, 7, 5, 0]

遞迴印出所有解

```
int row[MAXN], m;
void dfs(int y) {
  if (y == n) {
    print();
    return;
  }
  for (int x = 0; x < n; ++x) {
    if (!isValidQueenPosition(y, x))
      continue;
    update(y, x, true);
    row[m++] = x;
    dfs(y + 1);
    update(y, x, false);
    --m;
  }
}
```

不能放的區域能不能用二進位表示?



0	0	1	1	1	0	0	0

R

0	0	1	0	0	0	0	0

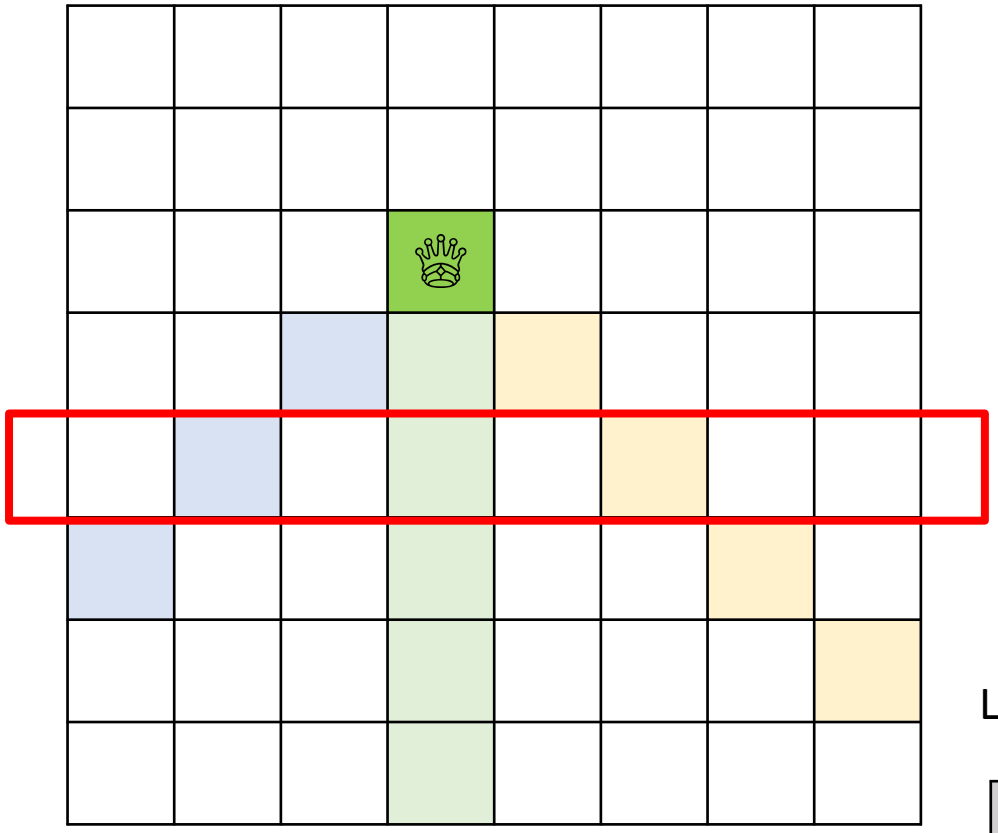
L

0	0	0	0	1	0	0	0

M

0	0	0	1	0	0	0	0

不能放的區域能不能用二進位表示?



0	1	0	1	0	1	0	0

R

0	1	0	0	0	0	0	0

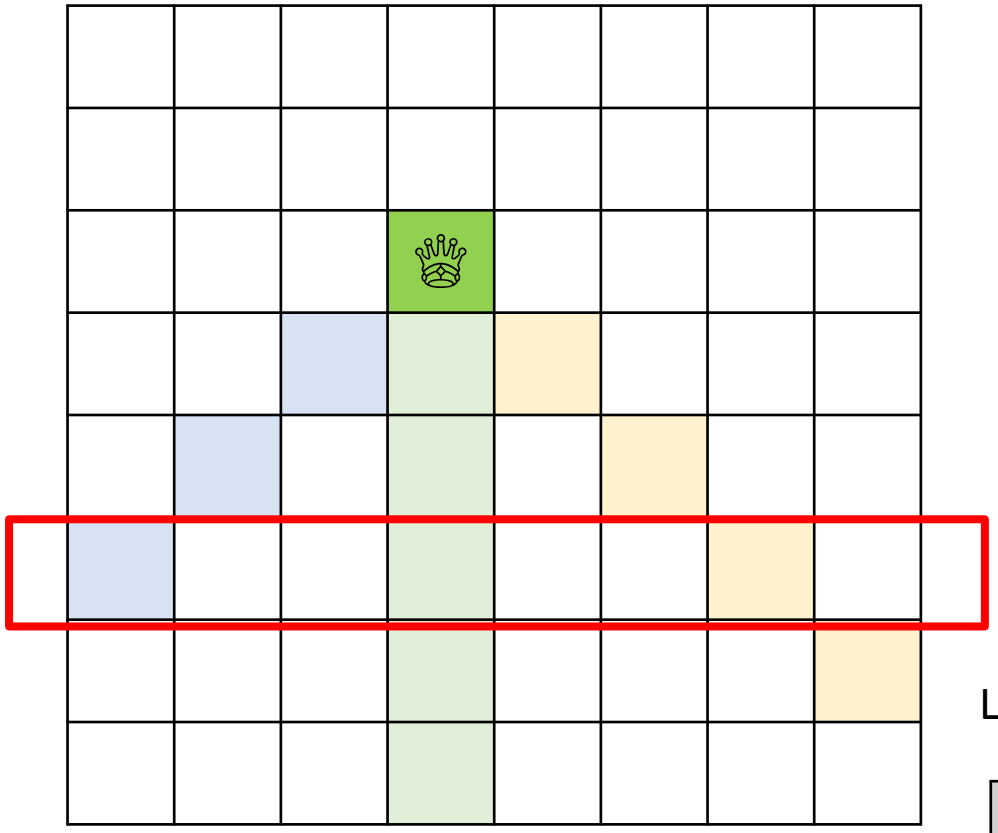
L

0	0	0	0	0	1	0	0

M

0	0	0	1	0	0	0	0

不能放的區域能不能用二進位表示?



1	0	0	1	0	0	1	0

R

1	0	0	0	0	0	0	0

L

0	0	0	0	0	0	1	0

M

0	0	0	1	0	0	0	0

利用 shift 操作

```cpp
#include <bitset>
#include <iostream>
using namespace std;

int main() {
  int L = 0, M = 0, R = 0;
  L = M = R = 1 << 12;
  for (int i = 0; i < 20; ++i) {
    cout << bitset<32>(L | M | R) <<
endl;
    L <<= 1;
    R >>= 1;
  }
  return 0;
}
```

```
00000000000000000001000000000000
00000000000000000111000000000000
00000000000000001010100000000000
00000000000000010010010000000000
00000000000000100010001000000000
00000000000001000010000100000000
00000000000010000010000010000000
00000000000100000010000001000000
00000000001000000010000000100000
00000000010000000010000000010000
00000000100000000010000000001000
00000001000000000010000000000100
00000010000000000010000000000010
00000100000000000010000000000001
00001000000000000010000000000000
00010000000000000010000000000000
00100000000000000010000000000000
01000000000000000010000000000000
10000000000000000010000000000000
```

位元運算加速

```cpp
#include <iostream>
using namespace std;

int MASK;
int ans;
void dfs(int M, int L, int R);

int main() {
  int n = 0;
  cin >> n;
  MASK = (1 << n) - 1;
  dfs(0, 0, 0);
  cout << ans << endl;
  return 0;
}
```

位元運算加速

```
int lowbit(int x) { return x & -x; }
void dfs(int M, int L, int R) {
  if (M == MASK) {
    ++ans;
    return;
  }
  int Legal = MASK & ~(M | L | R);
  for (int num = 0; Legal; Legal ^= num) {
    num = lowbit(Legal);
    dfs(M | num, (L | num) << 1, (R | num) >> 1);
  }
}
```