6.172
Performance
Engineering of
Software Systems

LECTURE 2
Bit Hacks

Charles E. Leiserson

September 14, 2010

© 2010 Charles E. Leiserson

Swap

Problem

Swap two integers x and y.

```
t = x;
x = y;
y = t;
```

© 2009 Charles E. Leiserson

No-Temp Swap

Problem

Swap two integers x and y without using a temporary.

```
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Example

x	10111101	10010011	10010011	00101110
y	00101110	00101110	10111101	10111101

No-Temp Swap

Problem

Swap two integers x and y without using a temporary.

```
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Example

x	10111101	**10010011**	10010011	00101110
y	00101110	00101110	10111101	10111101

No-Temp Swap

Problem

Swap two integers x and y without using a temporary.

```
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Example

x	10111101	10010011	10010011	00101110
y	00101110	00101110	10111101	10111101

© 2009 Charles E. Leiserson

No-Temp Swap

Problem

Swap two integers x and y without using a temporary.

```
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Example

x	10111101	10010011	10010011	**00101110**
y	00101110	00101110	10111101	10111101

No-Temp Swap

Problem

Swap two integers x and y without using a temporary.

```
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Example

x	10111101	10010011	10010011	00101110
y	00101110	00101110	10111101	10111101

Why it works

XOR is its own inverse: $(x \wedge y) \wedge y = x$.

Performance

Poor at exploiting instruction-level parallelism (ILP).

Minimum of Two Integers

Problem

Find the minimum r of two integers x and y.

```
if (x < y)
    r = x;
else
    r = y;
```

or

```
r = (x < y) ? x : y;
```

Performance

A mispredicted branch empties the processor pipeline

- ~16 cycles on the cloud facility's Intel Core i7's.

The compiler might be smart enough to avoid the unpredictable branch, but maybe not.

© 2009 Charles E. Leiserson

No–Branch Minimum

Problem

Find the minimum z of two integers x and y without a branch.

```
r = y ^ ((x ^ y) & -(x < y));
```

Why it works:

- C represents the Booleans TRUE and FALSE with the integers 1 and 0, respectively.
- If $x < y$, then $-(x < y) = -1$, which is all 1's in two's complement representation. Therefore, we have $y \wedge (x \wedge y) = x$.
- If $x \geq y$, then $-(x < y) = 0$. Therefore, we have $y \wedge 0 = y$.

Modular Addition

Problem

Compute $(x + y) \bmod n$, assuming that $0 \leq x < n$ and $0 \leq y < n$.

```
r = (x + y) % n;
```

Divide is expensive, unless by a power of 2.

```
z = x + y;
r = (z < n) ? z : z-n;
```

Unpredictable branch is expensive.

```
z = x + y;
r = z - (n & -(z >= n));
```

Same trick as minimum.

Round up to a Power of 2

Problem

Compute $2^{\lceil \log n \rceil}$.

```
//64-bit integers
--n;
n |= n >> 1;
n |= n >> 2;
n |= n >> 4;
n |= n >> 8;
n |= n >> 16;
n |= n >> 32;
++n;
```

Example

0010000001010000
0010000001001111
0011000001101111
0011110001111111
0011111111111111
0100000000000000

© 2009 Charles E. Leiserson

Round up to a Power of 2

Problem

Compute $2^{\lceil \log n \rceil}$.

```
//64-bit integers
--n;
n |= n >> 1;
n |= n >> 2;
n |= n >> 4;
n |= n >> 8;
n |= n >> 16;
n |= n >> 32;
++n;
```

Example

0010000001010000
0010000001001111
0011000001101111
0011110001111111
0011111111111111
0100000000000000

Round up to a Power of 2

Problem

Compute $2^{\lceil \log n \rceil}$.

```
//64-bit integers
--n;
n |= n >> 1;
n |= n >> 2;
n |= n >> 4;
n |= n >> 8;
n |= n >> 16;
n |= n >> 32;
++n;
```

Example

0010000001010000
0010000001001111
0011000001101111
0011110001111111
0011111111111111
0100000000000000

Round up to a Power of 2

Problem

Compute $2^{\lceil \log n \rceil}$.

```
//64-bit integers
--n;
n |= n >> 1;
n |= n >> 2;
n |= n >> 4;
n |= n >> 8;
n |= n >> 16;
n |= n >> 32;
++n;
```

Example

0010000001010000
0010000001001111
0011000001101111
0011110001111111
0011111111111111
0100000000000000

Round up to a Power of 2

Problem

Compute $2^{\lceil \log n \rceil}$.

```
//64-bit integers
--n;
n |= n >> 1;
n |= n >> 2;
n |= n >> 4;
n |= n >> 8;
n |= n >> 16;
n |= n >> 32;
++n;
```

Example

0010000001010000
0010000001001111
0011000001101111
0011110001111111
0011111111111111
0100000000000000

© 2009 Charles E. Leiserson

Round up to a Power of 2

Problem

Compute $2^{\lceil \log n \rceil}$.

```
//64-bit integers
--n;
n |= n >> 1;
n |= n >> 2;
n |= n >> 4;
n |= n >> 8;
n |= n >> 16;
n |= n >> 32;
++n;
```

Example

0010000001010000
0010000001001111
0011000001101111
0011110001111111
0011111111111111
0100000000000000

Round up to a Power of 2

Problem

Compute $2^{\lceil \log n \rceil}$.

```
//64-bit integers
--n;
n |= n >> 1;
n |= n >> 2;
n |= n >> 4;
n |= n >> 8;
n |= n >> 16;
n |= n >> 32;
++n;
```

Example

0010000001010000
0010000001001111
0011000001101111
0011110001111111
0011111111111111
0100000000000000

Why decrement and increment?

To handle the boundary case when n is a power of 2.

© 2009 Charles E. Leiserson

Least-Significant 1

Problem

Compute the mask of the least-significant 1 in word x.

```
r = x & (-x);
```

Example

x	0010000001010000
−x	1101111110110000
x & (−x)	0000000000010000

Question

How do you find the index of the bit, i.e., $\lg r = \log_2 r$?

Log Base 2 of a Power of 2

Problem

Compute lg x, where x is a power of 2.

```
const uint64_t deBruijn = 0x022fdd63cc95386d;
const unsigned int convert[64] =
{   0,  1,  2, 53,  3,  7, 54, 27,
    4, 38, 41,  8, 34, 55, 48, 28,
   62,  5, 39, 46, 44, 42, 22,  9,
   24, 35, 59, 56, 49, 18, 29, 11,
   63, 52,  6, 26, 37, 40, 33, 47,
   61, 45, 43, 21, 23, 58, 17, 10,
   51, 25, 36, 32, 60, 20, 57, 16,
   50, 31, 19, 15, 30, 14, 13, 12};

r = convert[(x*deBruijn) >> 58];
```

Log Base 2 of a Power of 2

Why it works

A ***deBruijn sequence*** s of length 2^k is a cyclic 0–1 sequence such that each of the 2^k 0–1 strings of length k occurs exactly once as a substring of s.

$00011101_2 * 2^4 = 11010000_2$
$11010000_2 >> 5 = 6$
convert[6] = 4

Performance

Limited by multiply and table look-up

Example k=3

00011101_2

0	000
1	001
2	011
3	111
4	110
5	101
6	010
7	100

convert[8] =
{0, 1, 6, 2, 7, 5, 4, 3};

Population Count I

Problem

Count the number of 1 bits in a word x.

```
for (r=0; x!=0; ++r)
  x &= x - 1;
```

Repeatedly eliminate the least-significant 1.

Example

x	0010110111010000
x−1	0010110111001111
x & (x−1)	0010110111000000

Issue

Fast if the popcount is small, but in the worst case, the running time is proportional to the number of bits in the word.

Population Count II

Table look-up

```
static const int count[256] =
  {0,1,1,2,1,2,2,3,1,...,8}; //#1's in index

for (r=0; x!=0; x>>=8)
  r += count[x & 0xFF];
```

Performance

Memory operations are much more costly than register operations:

- register: 1 cycle (6 ops issued per cycle per core),
- L1-cache: 4 cycles,
- L2-cache: 10 cycles,
- L3-cache: 50 cycles,
- DRAM: 150 cycles.

per 64-byte cache line

Population Count III

Parallel divide-and-conquer

Performance

$\Theta(\lg n)$ time, where n = word length.

```
// Create masks
B5 = !((-1) << 32);
B4 = B5 ^ (B5 << 16);
B3 = B4 ^ (B4 << 8);
B2 = B3 ^ (B3 << 4);
B1 = B2 ^ (B2 << 2);
B0 = B1 ^ (B1 << 1);
// Compute popcount
x = ((x >> 1) & B0) + (x & B0);
x = ((x >> 2) & B1) + (x & B1);
x = ((x >> 4) + x) & B2;
x = ((x >> 8) + x) & B3;
x = ((x >> 16) + x) & B4;
x = ((x >> 32) + x) & B5;
```

Population Count III

1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0

1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0

© 2009 Charles E. Leiserson

Population Count III

1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0

	1	1	1	1	0	1	0	0	0	1	1	1	1	0	0	0
+	1	1	0	0	0	0	1	0	0	1	0	1	1	0	1	1
	10	10	01	01	00	01	01	00	00	10	01	10	10	00	01	01

Population Count III

11110101000110000011011111001010

	1	1	1	1	0	1	0	0	0	1	1	1	1	0	0	0
+	1	1	0	0	0	0	1	0	0	1	0	1	1	0	1	1
	10	10	01	01	00	01	01	00	00	10	01	10	10	00	01	01

Population Count III

```
     1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0

       1   1   1   1   0   1   0   0   0   1   1   1   1   0   0   0
 +     1   1   0   0   0   0   1   0   0   1   0   1   1   0   1   1
 -----------------------------------------------------------------
         1 0     0 1     0 1     0 0     1 0     1 0     0 0     0 1
 +       1 0     0 1     0 0     0 1     0 0     0 1     1 0     0 1
 -----------------------------------------------------------------
     0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0
```

Population Count III

```
     1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0

       1   1   1   1   0   1   0   0   0   1   1   1   1   0   0   0
 +   1   1   0   0   0   0   1   0   0   1   0   1   1   0   1   1
 -----------------------------------------------------------------
   |     1 0 |     0 1 |     0 1 |     0 0 |     1 0 |     1 0 |     0 0 |     0 1 |
 + |     1 0 |     0 1 |     0 0 |     0 1 |     0 0 |     0 1 |     1 0 |     0 1 |
 -----------------------------------------------------------------
   | 0 1 0 0 | 0 0 1 0 | 0 0 0 1 | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | 0 0 1 0 | 0 0 1 0 |
```

Population Count III

```
    11110101000110000011011111001010

      1   1   1   1   0   1   0   0   0   1   1   1   1   0   0   0
  +   1   1   0   0   0   0   1   0   0   1   0   1   1   0   1   1
  ----------------------------------------------------------------
         10      01      01      00      10      10      00      01
  +      10      01      00      01      00      01      10      01
  ----------------------------------------------------------------
    |        0010|        0001|        0011|        0010|
  + |        0100|        0001|        0010|        0010|
  ----------------------------------------------------------------
    |00000110    |00000010    |00000101    |00000100    |
```

Population Count III

```
    11110101000110000011011111001010

      1  1  1  1  0  1  0  0  0  1  1  1  1  0  0  0
 +    1  1  0  0  0  0  1  0  0  1  0  1  1  0  1  1
 ------------------------------------------------------
         10    01    01    00    10    10    00    01
 +       10    01    00    01    00    01    10    01
 ------------------------------------------------------
           0010        0001        0011        0010
 +         0100        0001        0010        0010
 ------------------------------------------------------
     00000110    00000010    00000101    00000100
```

Population Count III

```
  11110101000110000011011111001010

    1   1   1   1   0   1   0   0   0   1   1   1   1   0   0   0
+   1   1   0   0   0   0   1   0   0   1   0   1   1   0   1   1
----------------------------------------------------------------
      10      01      01      00      10      10      00      01
+     10      01      00      01      00      01      10      01
----------------------------------------------------------------
          0010            0001            0011            0010
+         0100            0001            0010            0010
----------------------------------------------------------------
              00000010                        00000100
+             00000110                        00000101
----------------------------------------------------------------
  00000000000000000000100000000000000000000000001001
```

Population Count III

```
 11110101000110000011011111001010

   1  1  1  1  0  1  0  0  0  1  1  1  1  0  0  0
+  1  1  0  0  0  0  1  0  0  1  0  1  1  0  1  1
---------------------------------------------------
     10    01    01    00    10    10    00    01
+    10    01    00    01    00    01    10    01
---------------------------------------------------
        0010        0001        0011        0010
+       0100        0001        0010        0010
---------------------------------------------------
            00000010                00000100
+           00000110                00000101
---------------------------------------------------
 0000000000001000 0000000000001001
```

Population Count III

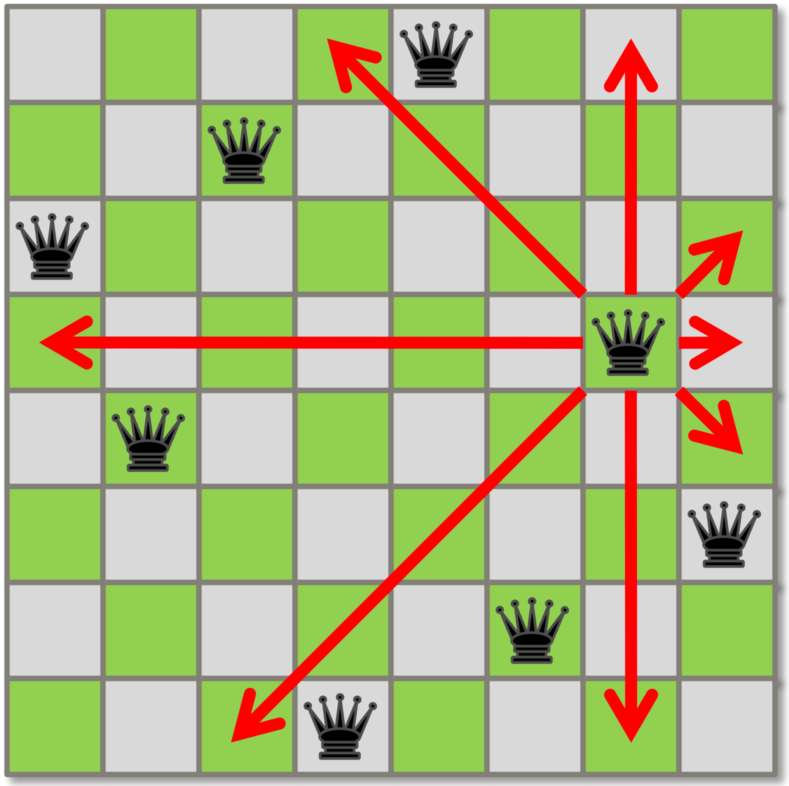
```
  1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0

    1   1   1   1   0   1   0   0   0   1   1   1   1   0   0   0
+   1   1   0   0   0   0   1   0   0   1   0   1   1   0   1   1
-----------------------------------------------------------------
      1 0     0 1     0 1     0 0     1 0     1 0     0 0     0 1
+     1 0     0 1     0 0     0 1     0 0     0 1     1 0     0 1
-----------------------------------------------------------------
          0 0 1 0         0 0 0 1         0 0 1 1         0 0 1 0
+         0 1 0 0         0 0 0 1         0 0 1 0         0 0 1 0
-----------------------------------------------------------------
                  0 0 0 0 0 0 1 0                 0 0 0 0 0 1 0 0
+                 0 0 0 0 0 1 1 0                 0 0 0 0 0 1 0 1
-----------------------------------------------------------------
                                  0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
+                                 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
-----------------------------------------------------------------
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
                                                        \_______/
                                                           17
```

Queens Problem

Problem

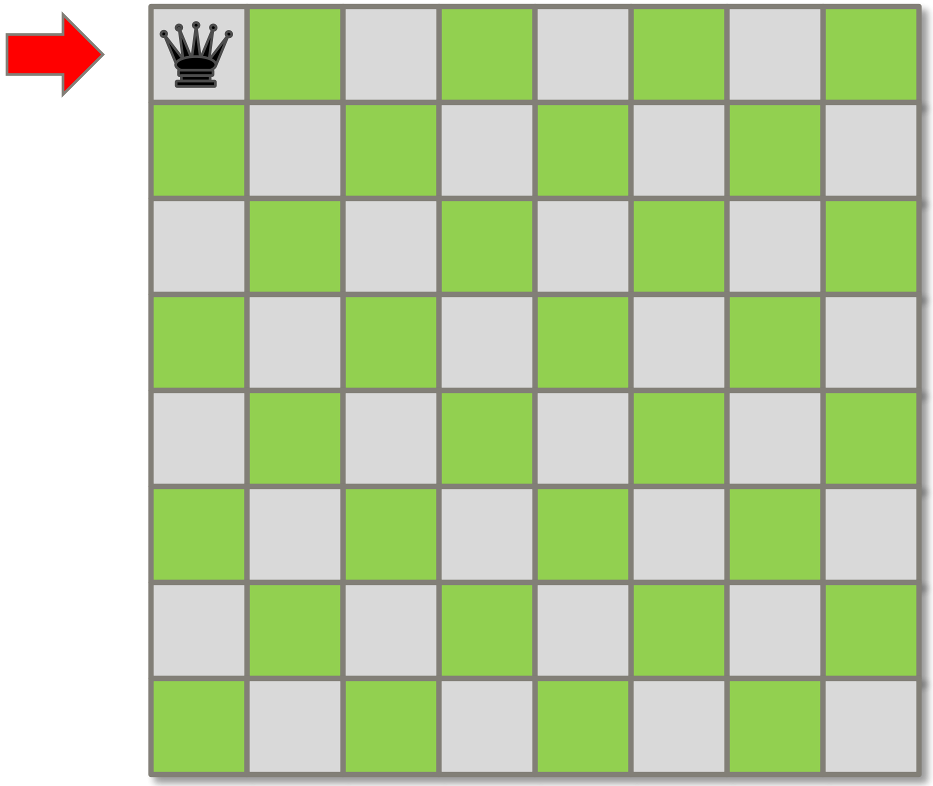
Place n queens on an n × n chessboard so that no queen attacks another, i.e., no two queens in any row, column, or diagonal.



Backtracking Search

Strategy

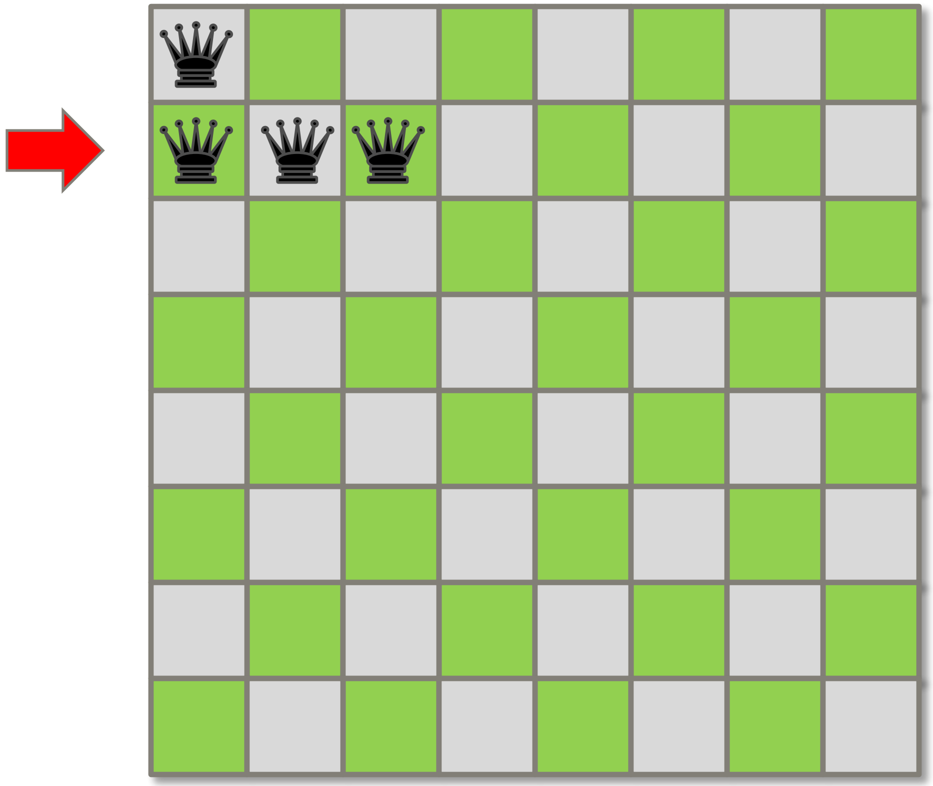
Try placing queens row by row. If you can't place a queen in a row, backtrack.



Backtracking Search

Strategy

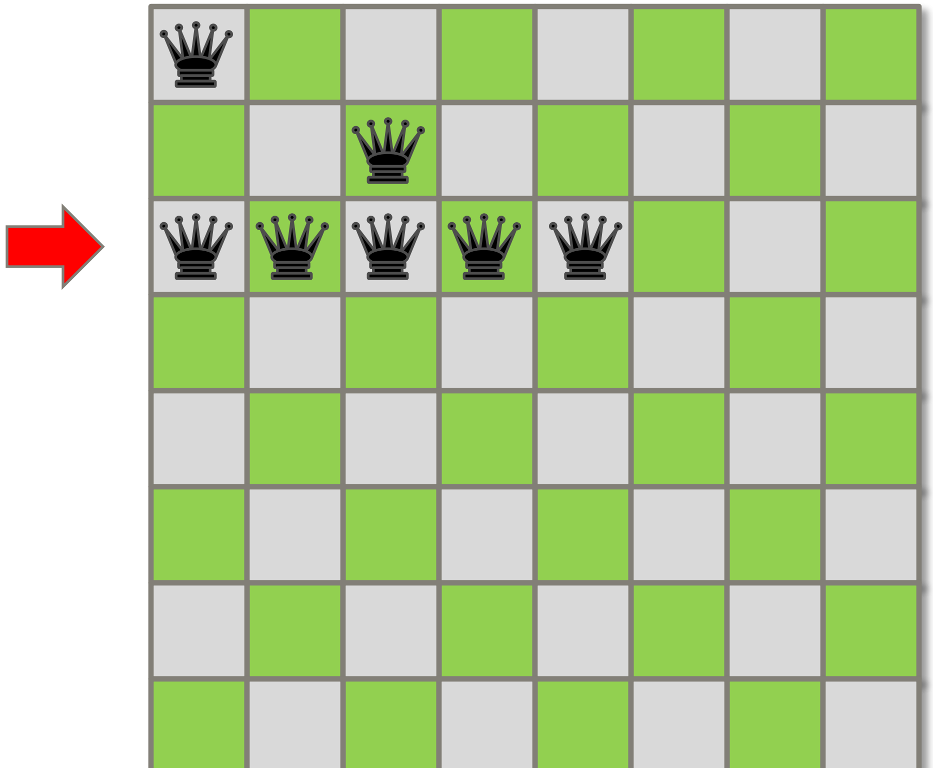
Try placing queens row by row. If you can't place a queen in a row, backtrack.



Backtracking Search

Strategy

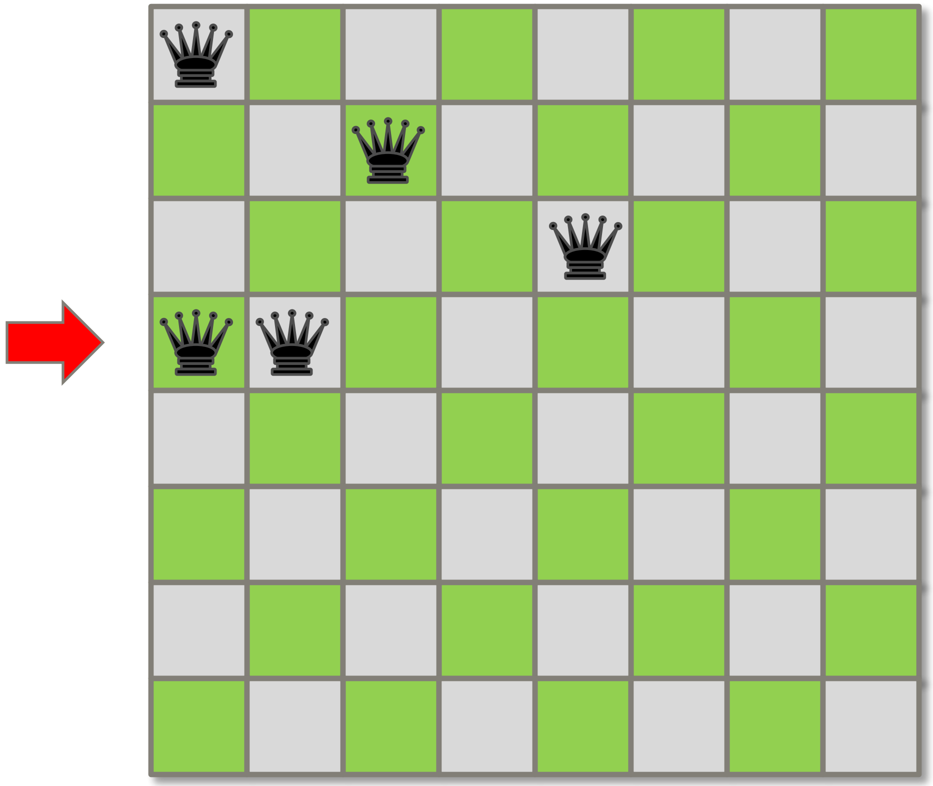
Try placing queens row by row. If you can’t place a queen in a row, backtrack.



Backtracking Search

Strategy

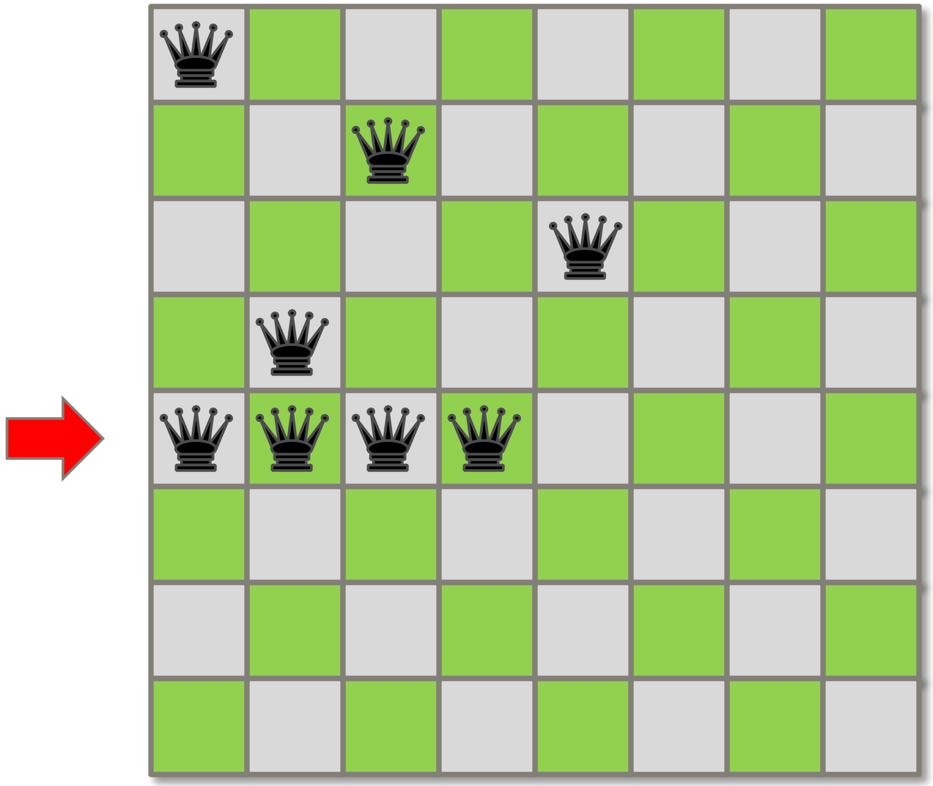
Try placing queens row by row. If you can't place a queen in a row, backtrack.



Backtracking Search

Strategy

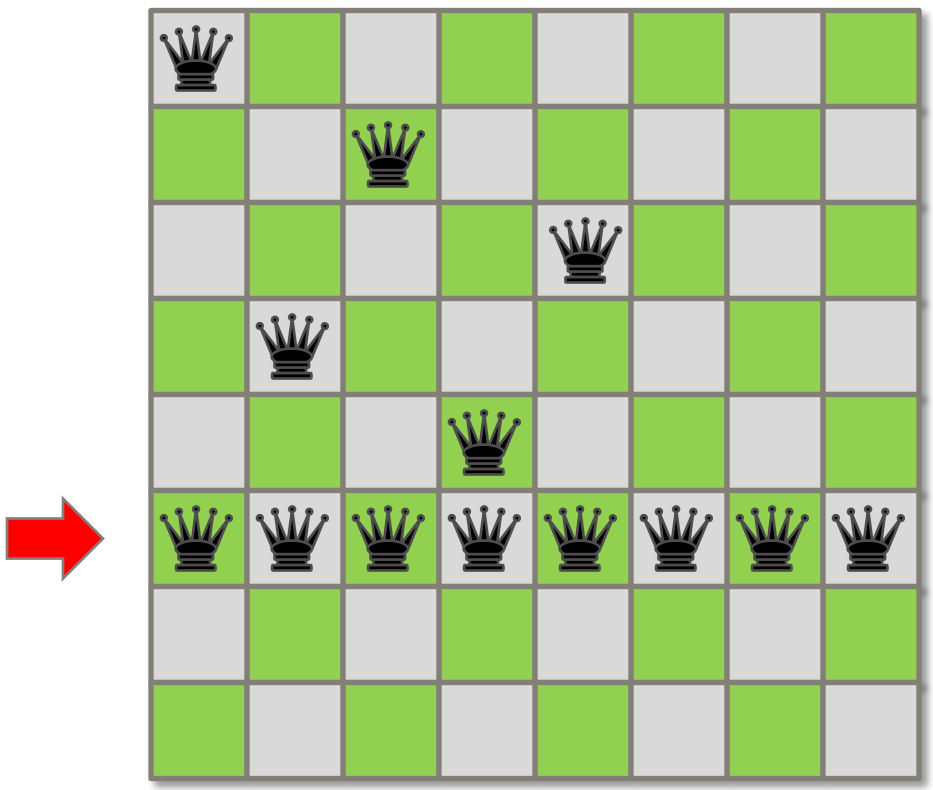
Try placing queens row by row. If you can't place a queen in a row, backtrack.



Backtracking Search

Strategy

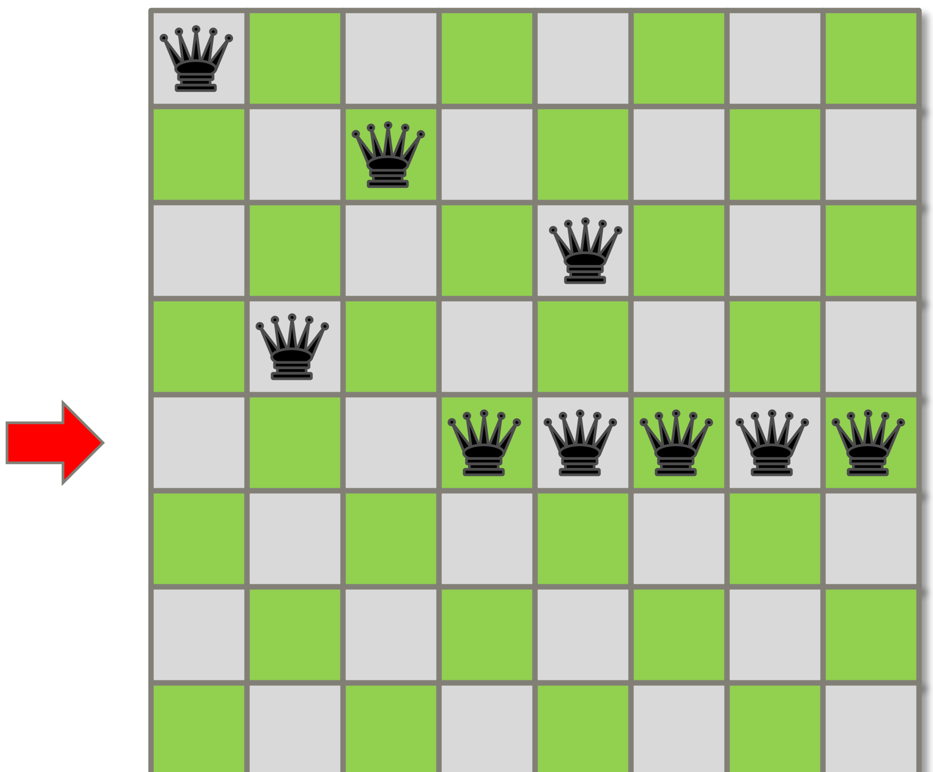
Try placing queens row by row. If you can't place a queen in a row, backtrack.



Backtracking Search

Strategy

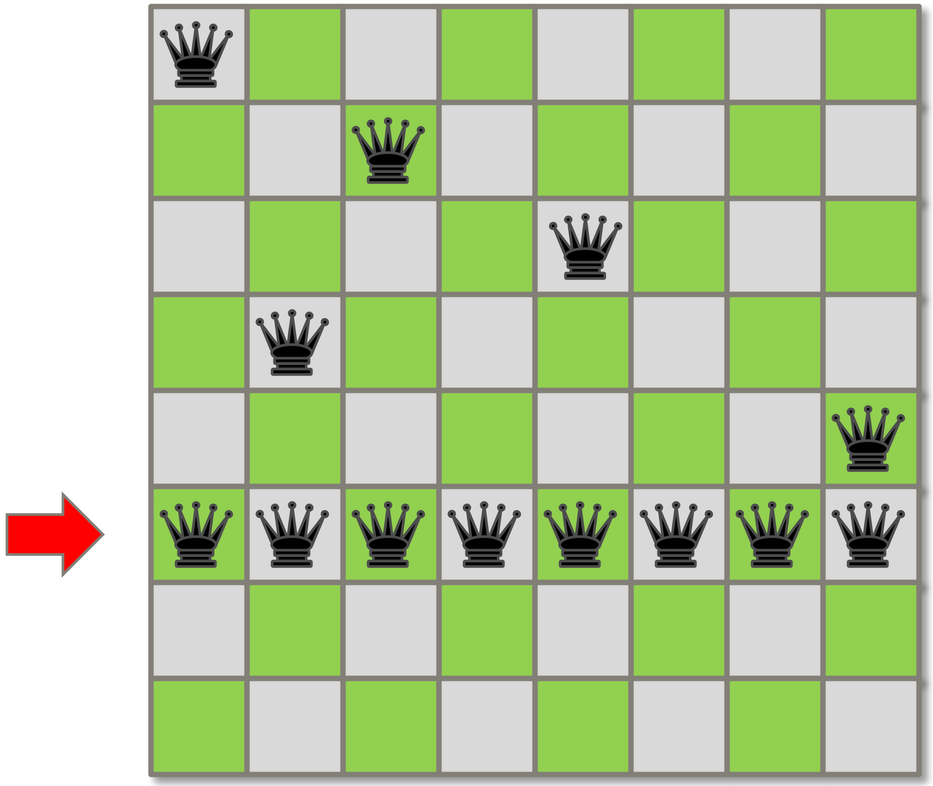
Try placing queens row by row. If you can't place a queen in a row, backtrack.



Backtracking Search

Strategy

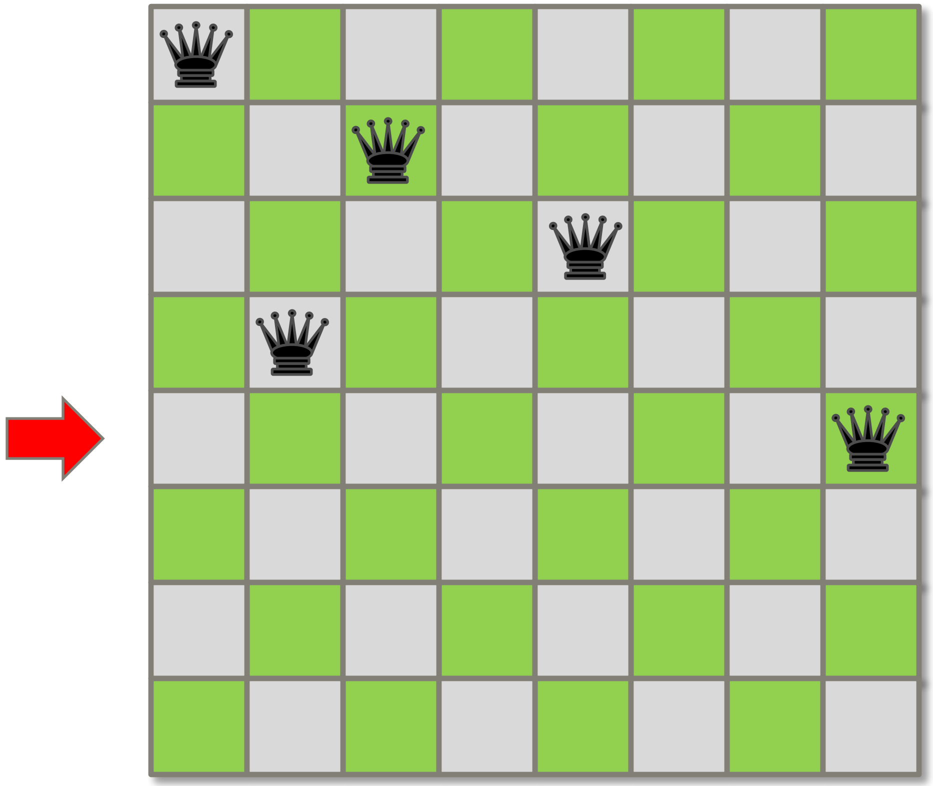
Try placing queens row by row. If you can't place a queen in a row, backtrack.



Backtracking Search

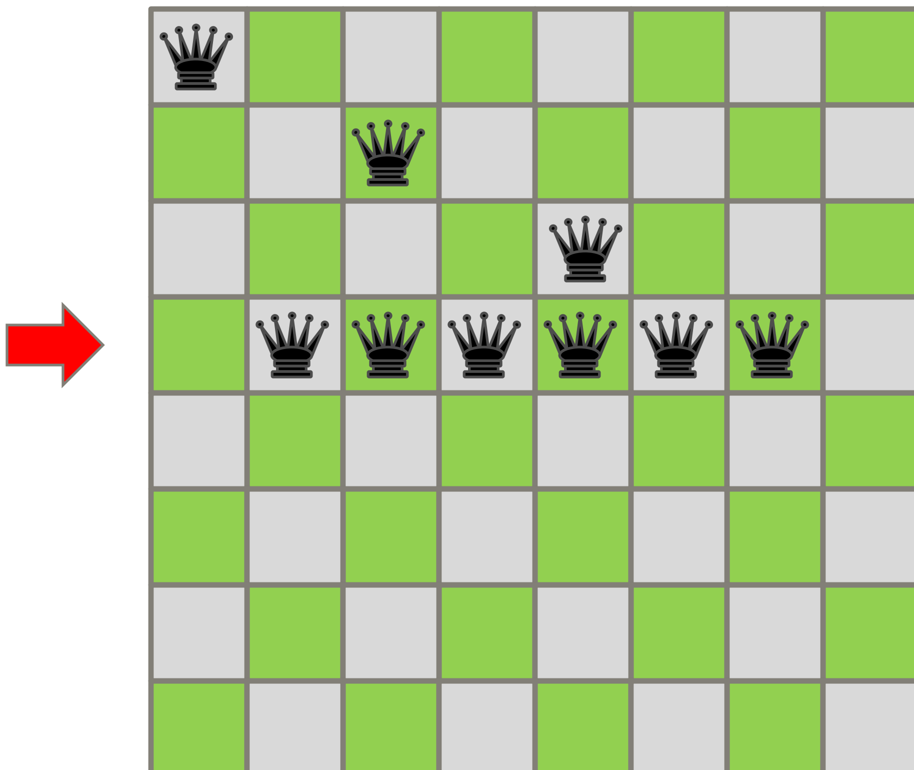
Strategy

Try placing queens row by row. If you can't place a queen in a row, backtrack.



Backtracking Search

Strategy
Try placing queens row by row. If you can't place a queen in a row, backtrack.



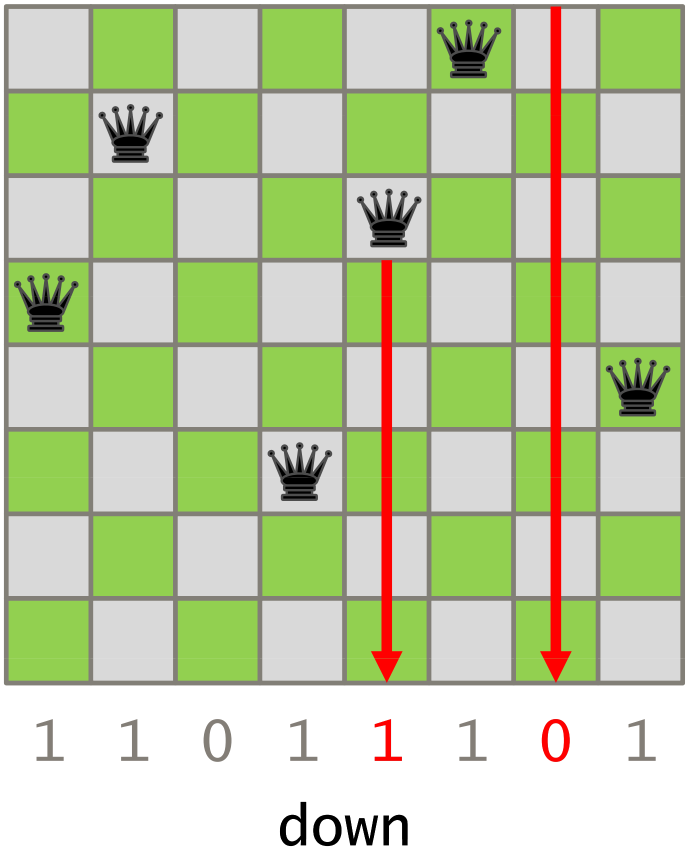
Board Representation

The backtrack search can be implemented as a simple recursive procedure, but how should the board be represented to facilitate queen placement?

- array of n^2 bytes?
- array of n^2 bits?
- array of n bytes?
- 3 bitvectors of size n, $2n-1$, and $2n-1$!

© 2009 Charles E. Leiserson

Bitvector Representation

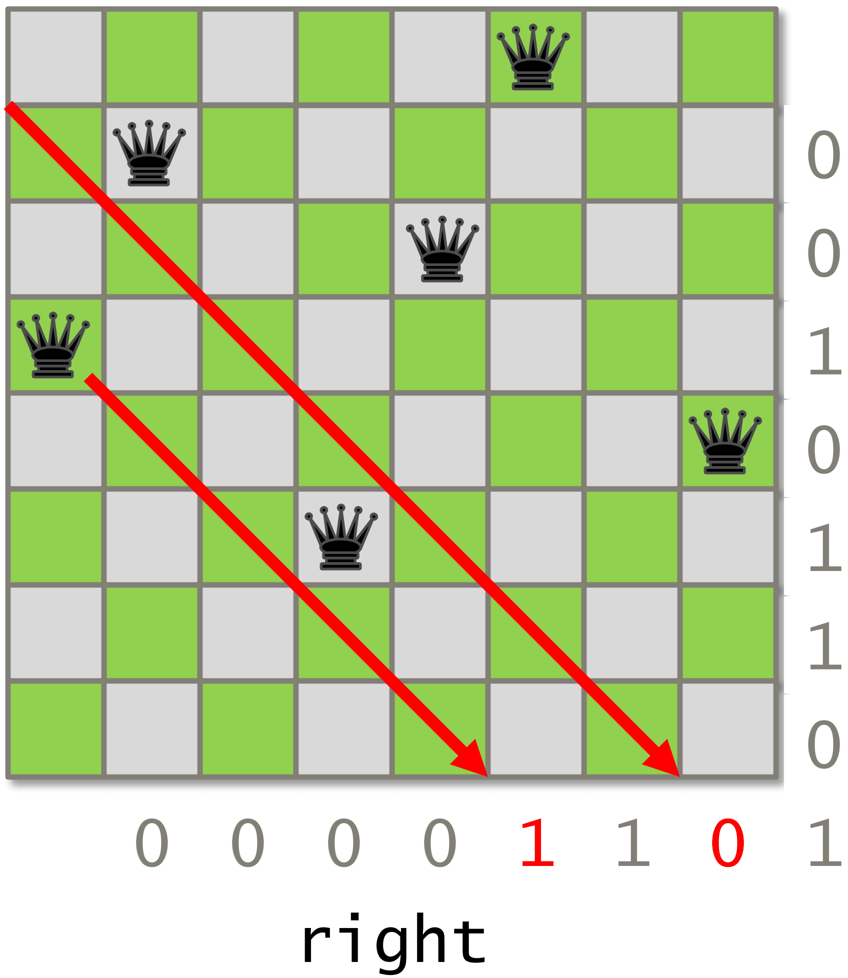


Placing a queen in column c is not safe if

```
down & (1<<c))
```

is nonzero.

Bitvector Representation



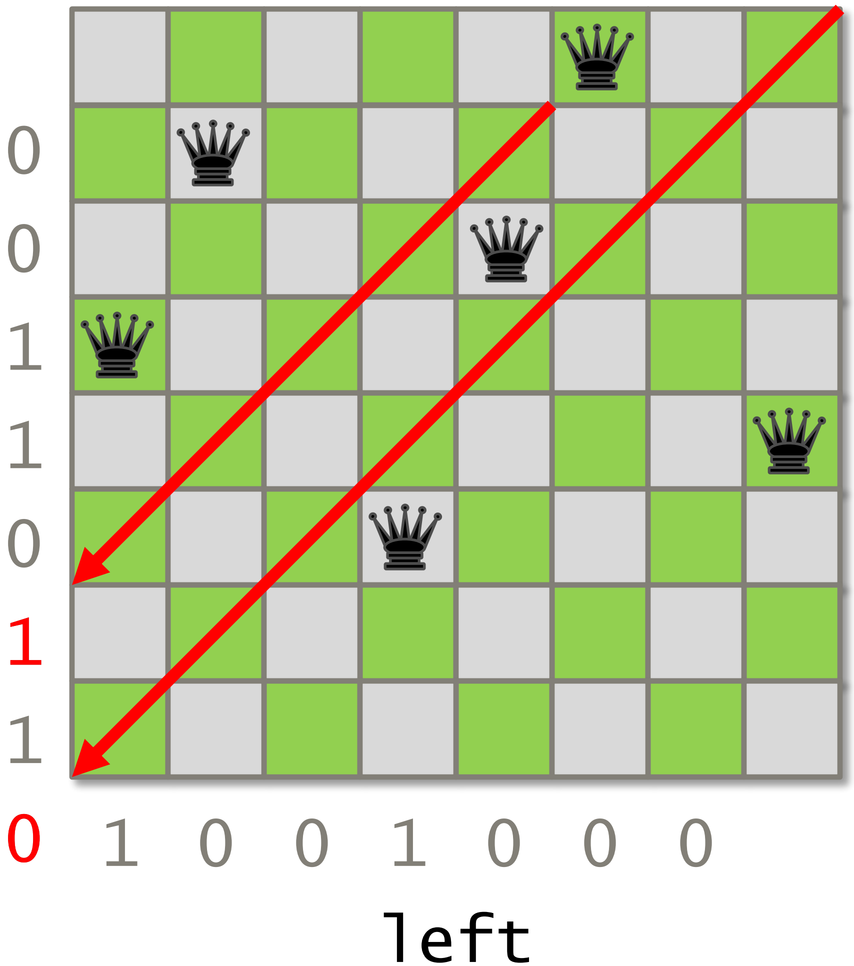
Placing a queen in row `r` and column `c` is not safe if

```
right & (1<<(n-r+c))
```

is nonzero.

© 2009 Charles E. Leiserson

Bitvector Representation



Placing a queen in row `r` and column `c` is not safe if

`left & (1<<(r+c))`

is nonzero.

Further Reading

Sean Eron Anderson, "Bit twiddling hacks," http://graphics.stanford.edu/~seander/bithacks.html, 2009.

Happy Hacking!

MIT OpenCourseWare
http://ocw.mit.edu

6.172 Performance Engineering of Software Systems
Fall 2010

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.