RECITATION 1.3 (SIT NEAR YOUR PROJECT MATES)

Announcements

1. Project 1Final Grading by Contribution

2. Homework 4 has a check-off today (Check-off 4 will be checked primarily)

3. After a short overview, I will go around asking each team (so each team gets about 5 minutes max.) about difficulties in the project and approaches used.

4. Feel free to leave after showing me the check-off and talking to me about the project!

Project Grading by Contribution

Final report should include the following:

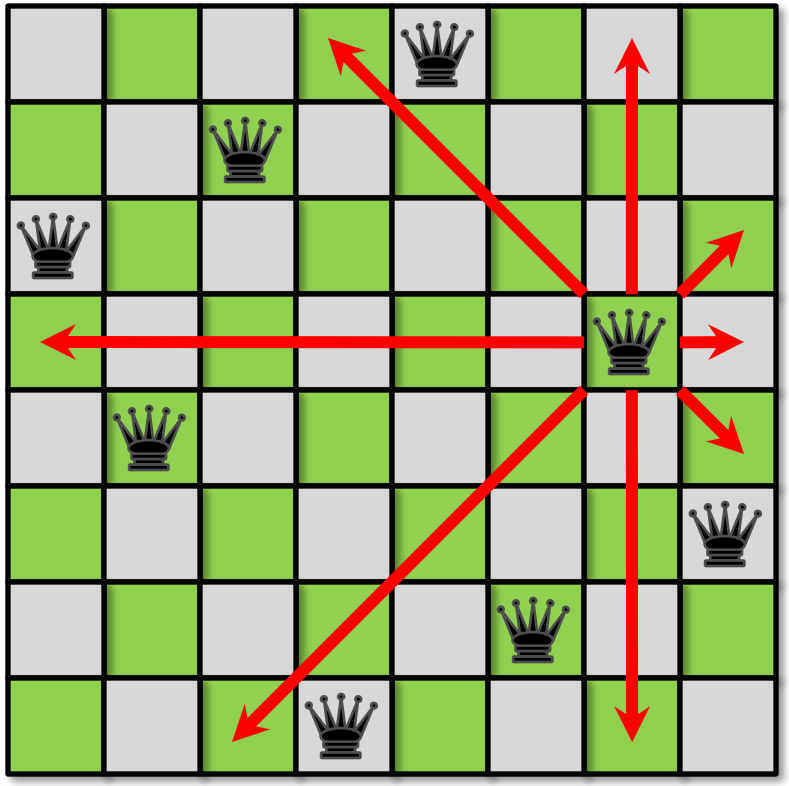
1. A clear project log on the various techniques each teammate explored and its outcomes.
2. Lessons learnt from an approach and the tier each approach reached.
3. With the above information, we will evaluate if one student has not contributed enough (we would like to not see it this way, but some cases are hard to look past…)
4. It is fine to not have one teammate's approach as the final Git submission, but the fact that they contributed to reach a significant tier through their alternative approach also carries weight.

Max Bezzel

The Queens Problem

Queens Problem

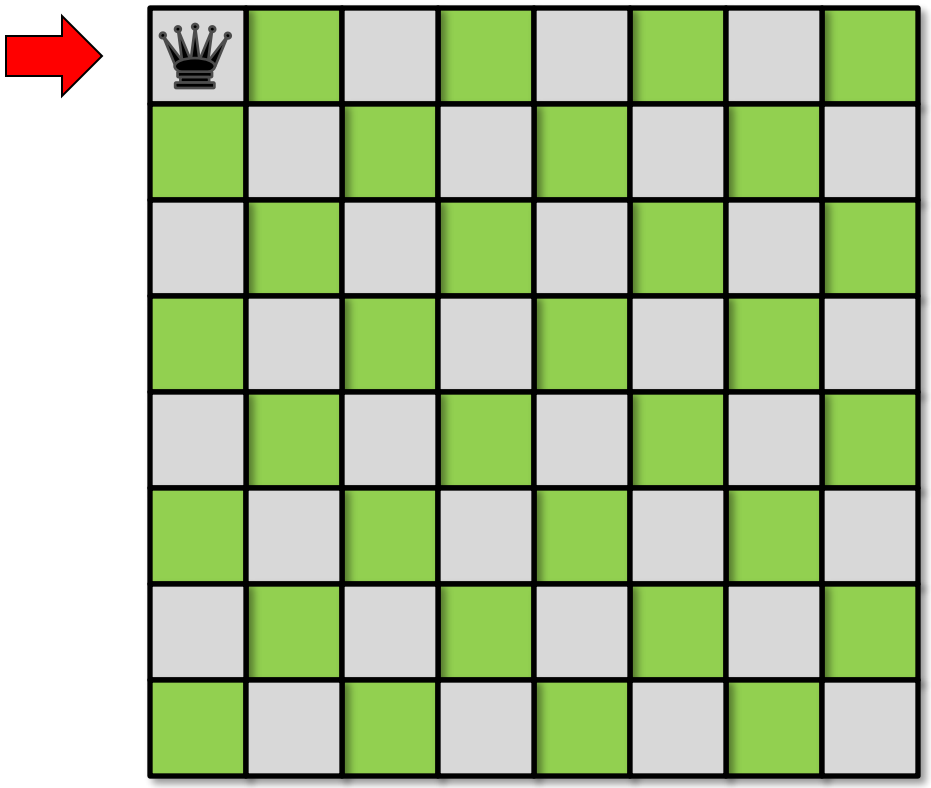
Place n Queens on an $n \times n$ chessboard so that no Queen attacks another, i.e., no two Queens in any row, column, or diagonal. Count the number of possible solutions.



Backtracking Search

Strategy

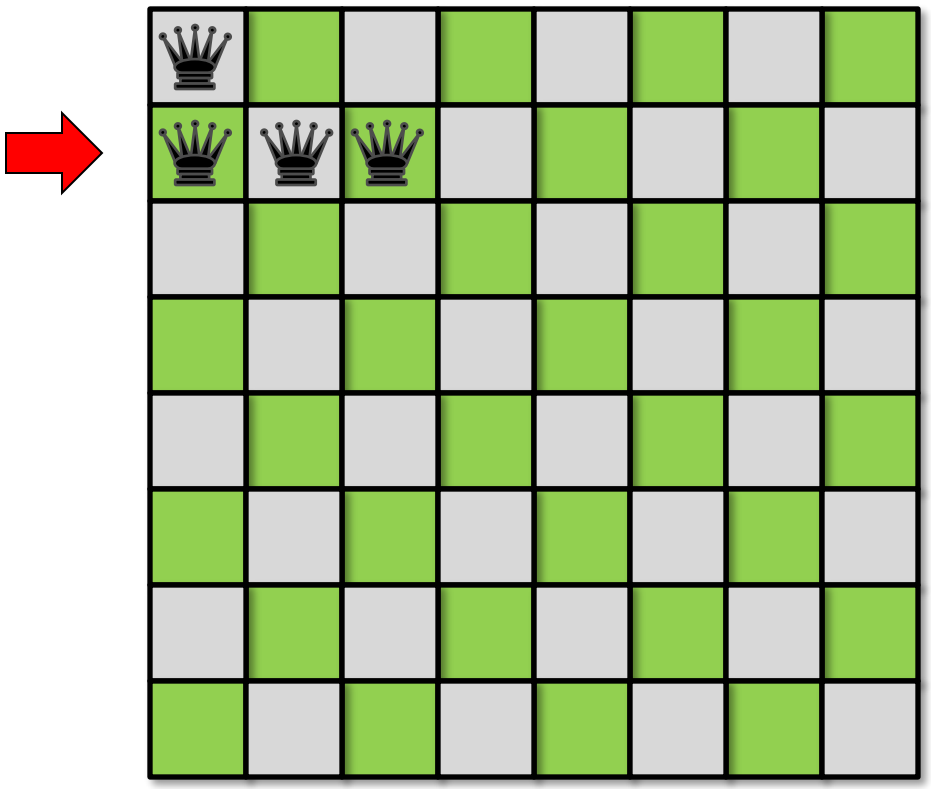
Try placing Queens row by row. If you can't place a Queen in a row, backtrack.



Backtracking Search

Strategy

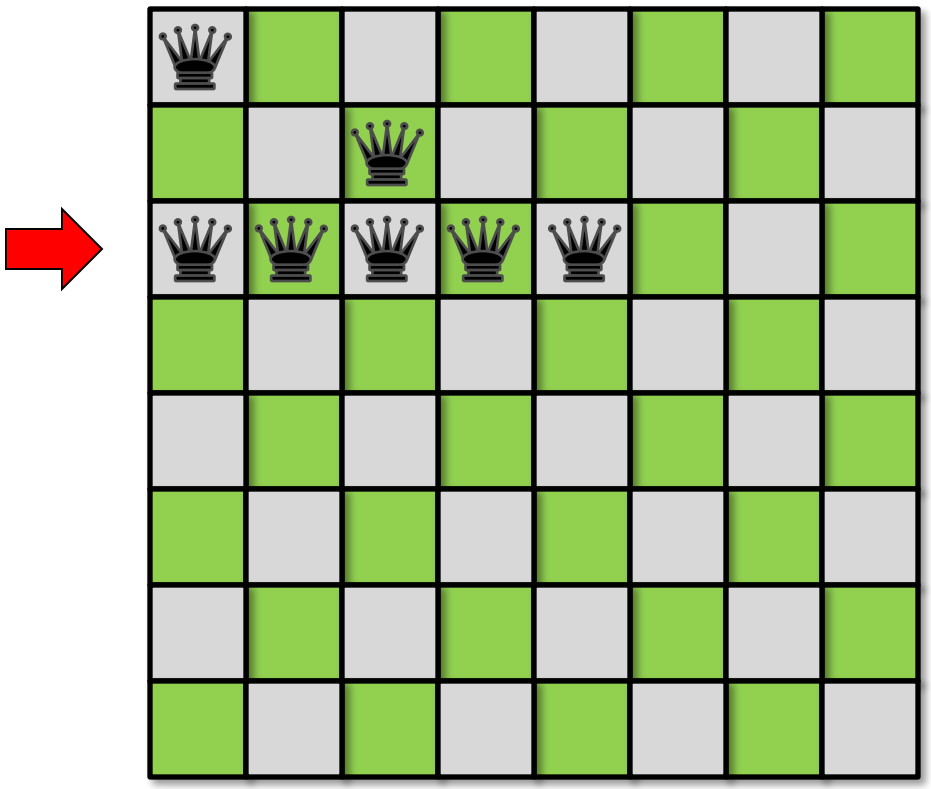
Try placing Queens row by row. If you can't place a Queen in a row, backtrack.



Backtracking Search

Strategy

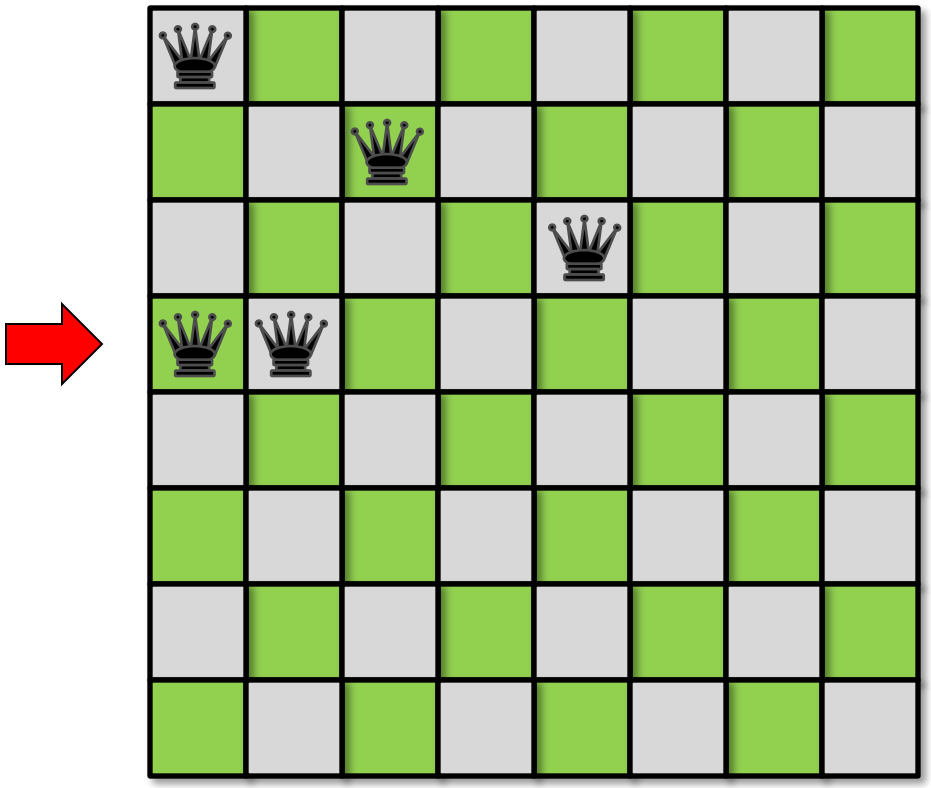
Try placing Queens row by row. If you can't place a Queen in a row, backtrack.



Backtracking Search

Strategy

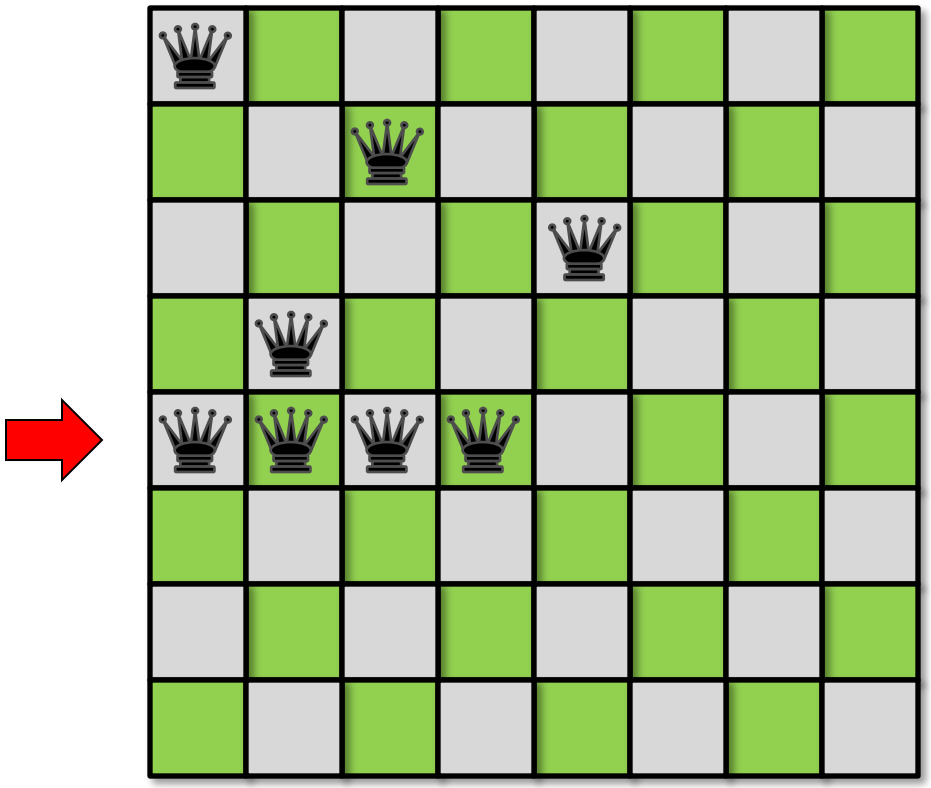
Try placing Queens row by row. If you can't place a Queen in a row, backtrack.



Backtracking Search

Strategy

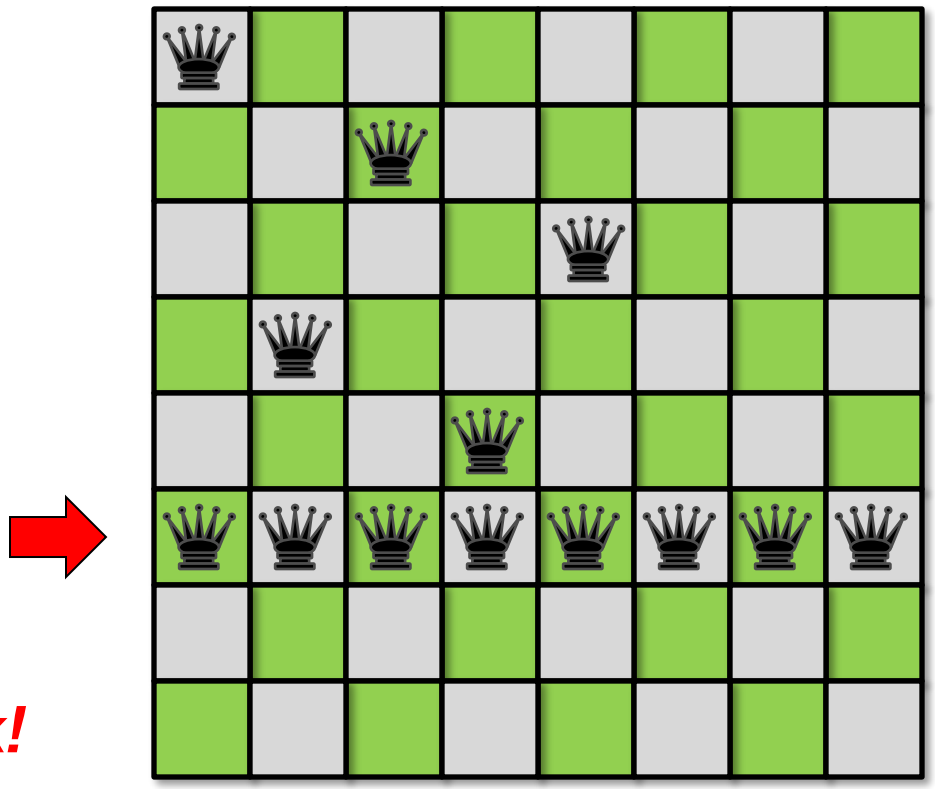
Try placing Queens row by row. If you can't place a Queen in a row, backtrack.



Backtracking Search

Strategy

Try placing Queens row by row. If you can't place a Queen in a row, backtrack.

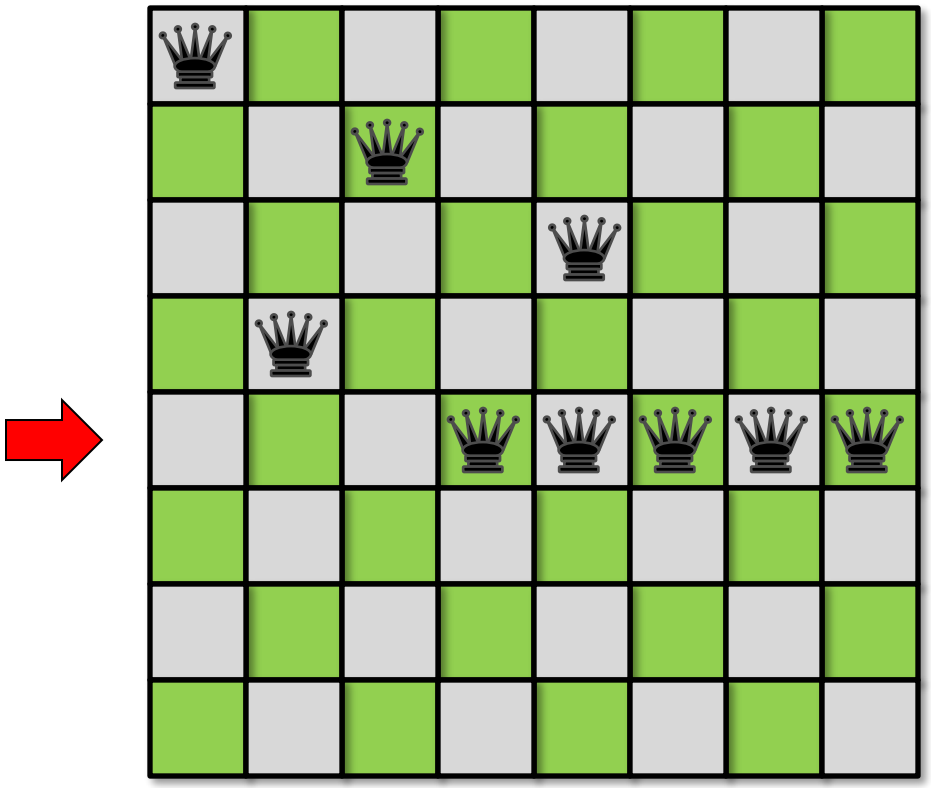


Backtrack!

Backtracking Search

Strategy

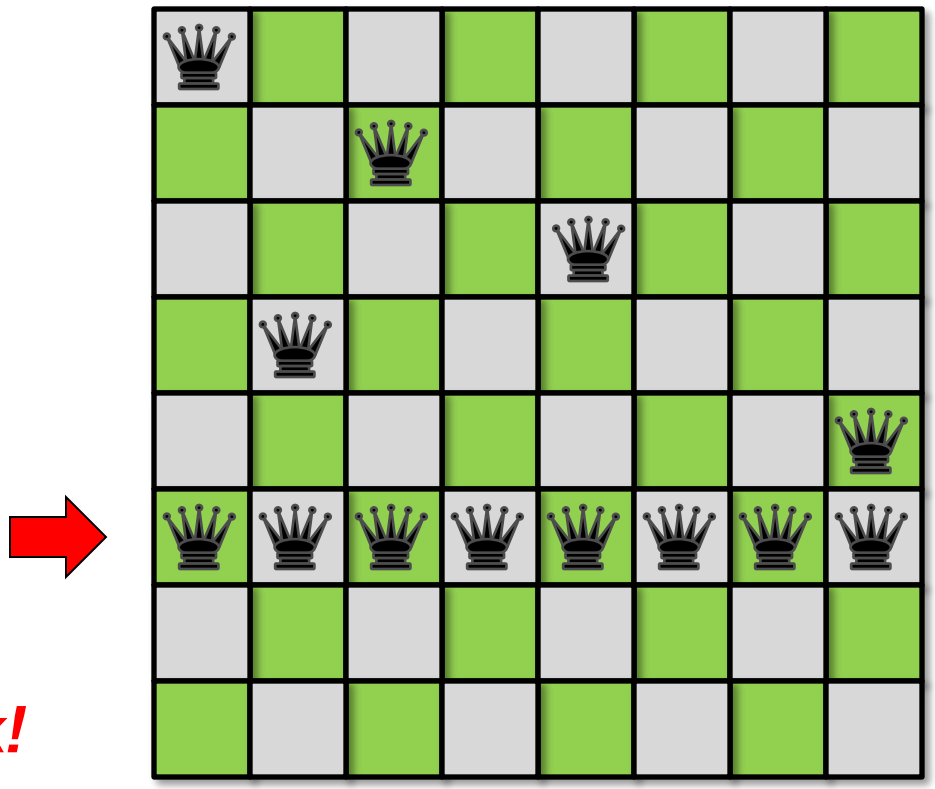
Try placing Queens row by row. If you can't place a Queen in a row, backtrack.



Backtracking Search

Strategy

Try placing Queens row by row. If you can't place a Queen in a row, backtrack.

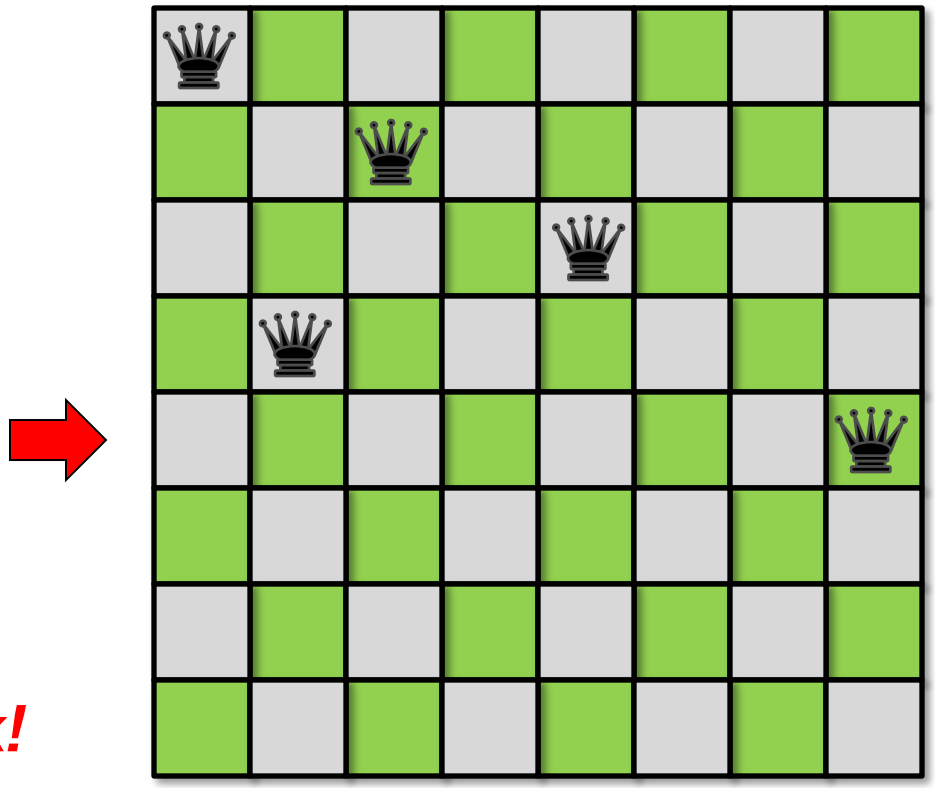


Backtrack!

Backtracking Search

Strategy

Try placing Queens row by row. If you can't place a Queen in a row, backtrack.

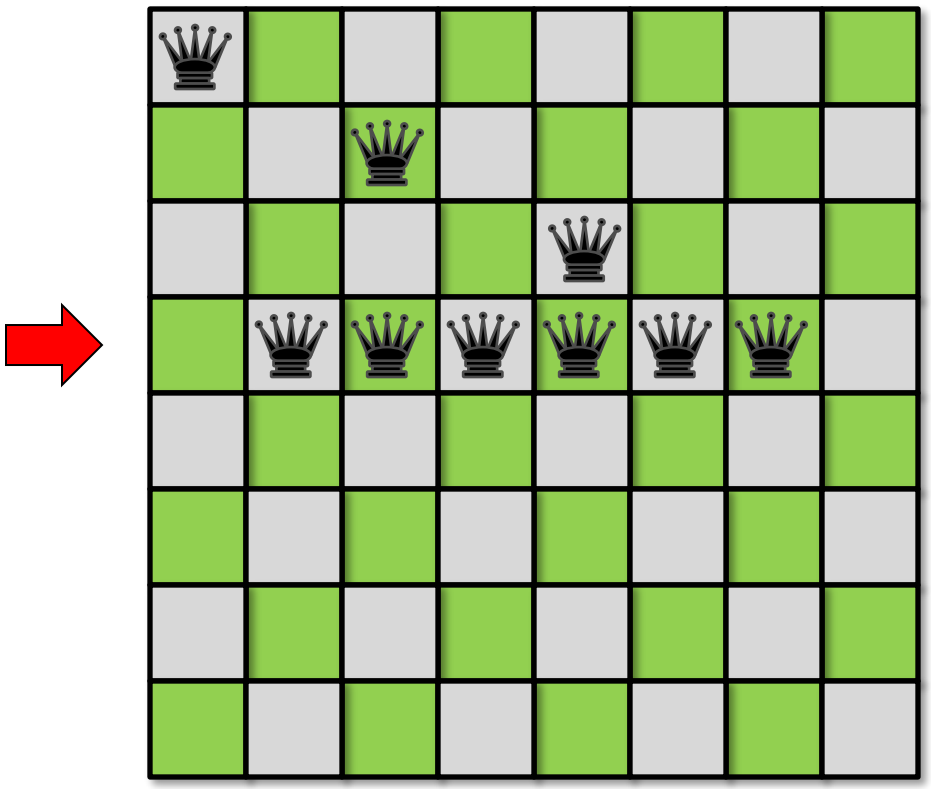


Backtrack!

Backtracking Search

Strategy

Try placing Queens row by row. If you can't place a Queen in a row, backtrack.

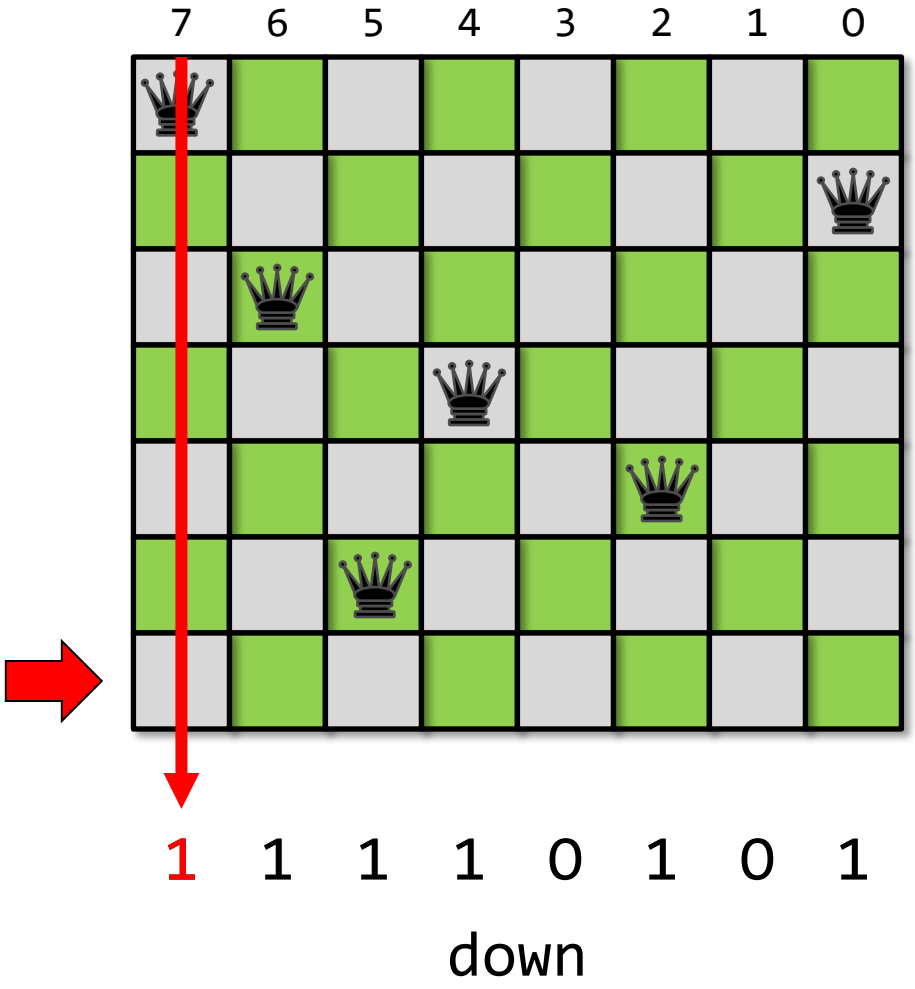


Board Representation

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.

Bitvector Representation

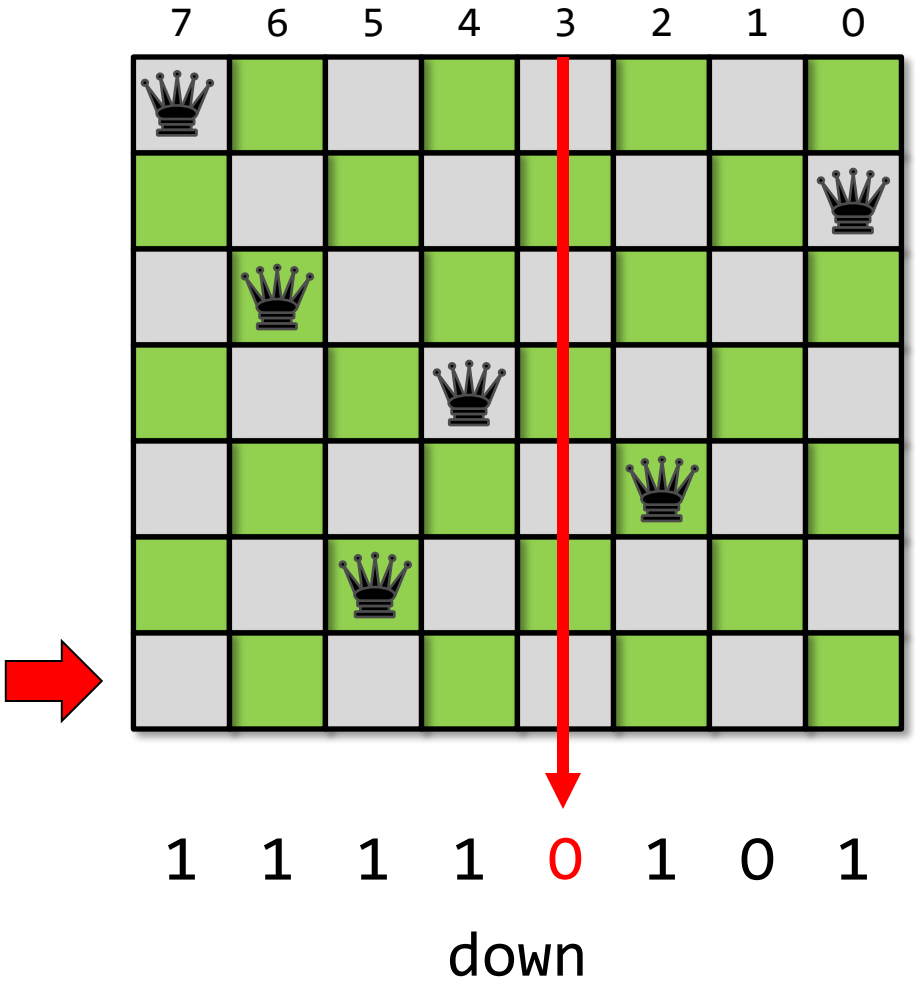


Placing a Queen in column `c` is not safe if

`down & place != 0`

where `place = 1<<c`.

Bitvector Representation

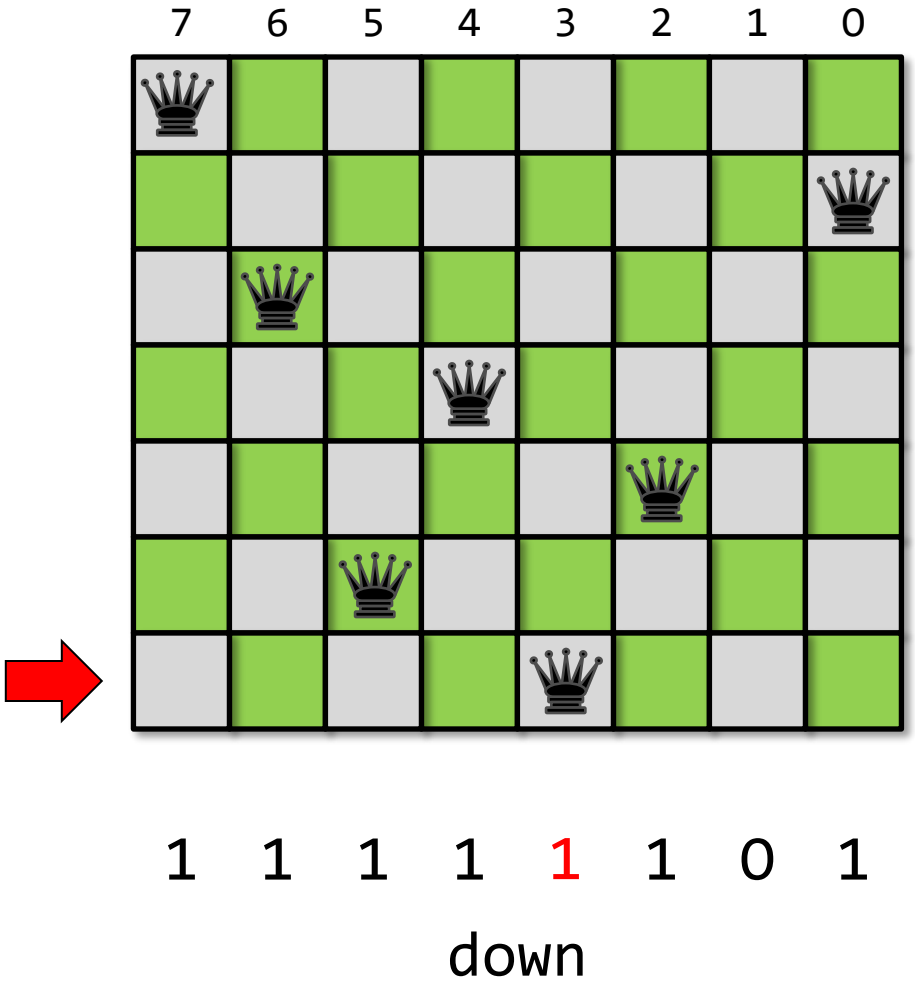


Placing a Queen in column `c` is not safe if

`down & place != 0`

where `place = 1<<c`.

Bitvector Representation



Placing a Queen in column `c` is not safe if

```
down & place != 0
```

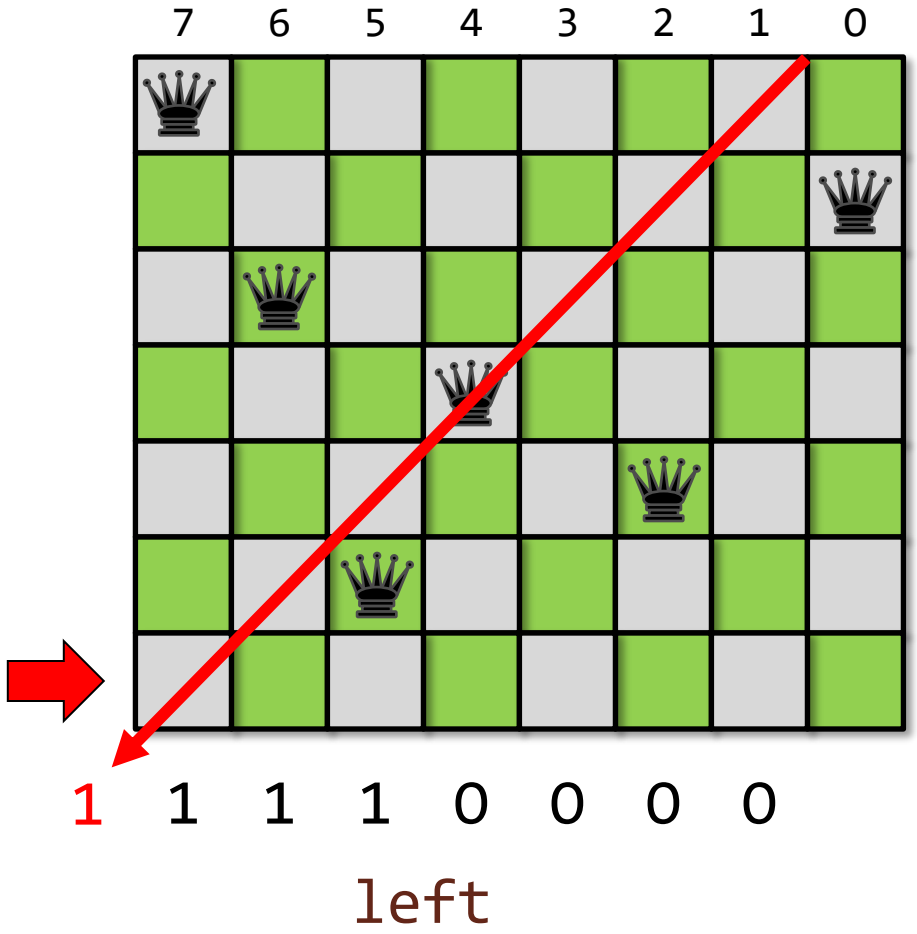
where `place = 1<<c`.

If column `c` is safe, then update

```
down |= place
```

for the next row.

Bitvector Representation

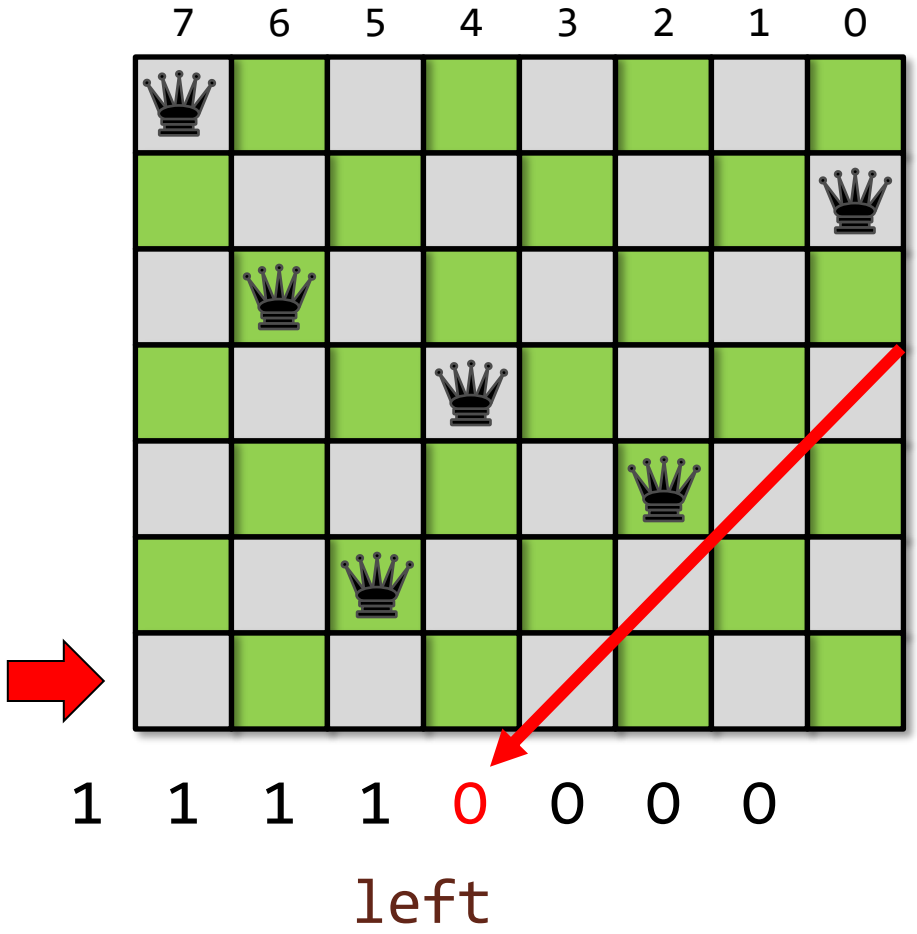


Placing a Queen in column `c` is not safe if

`left & place != 0`

where `place = 1<<c`.

Bitvector Representation

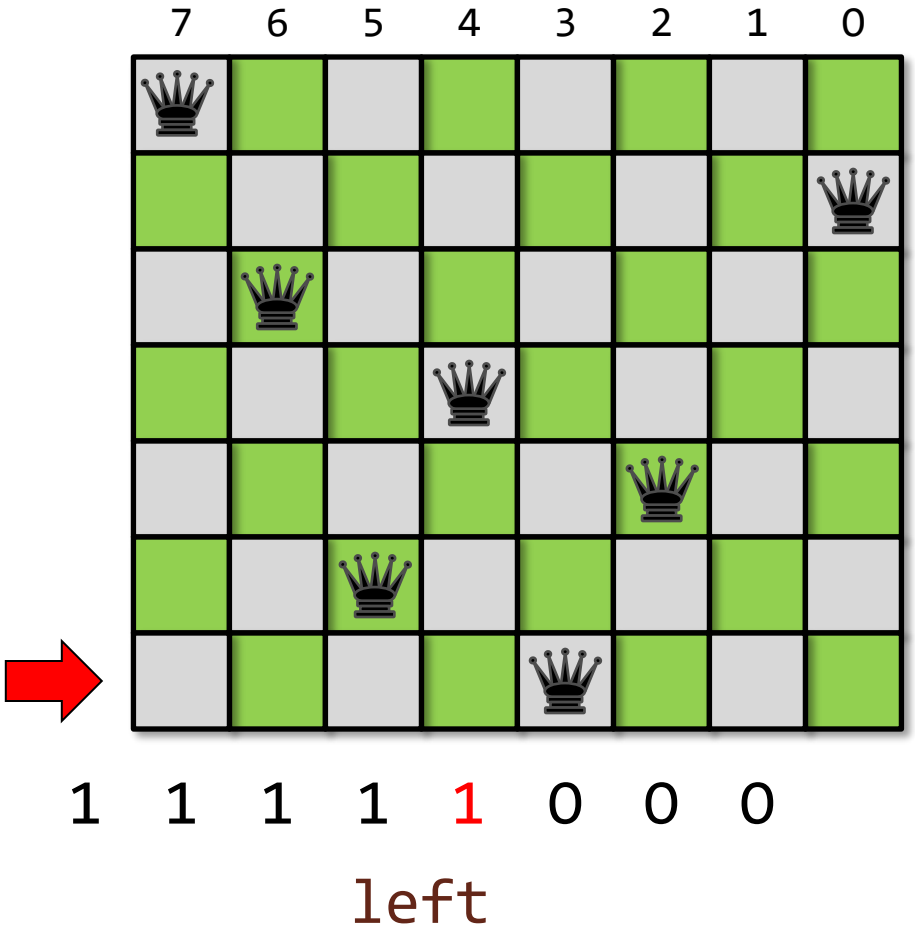


Placing a Queen in column `c` is not safe if

`left & place != 0`

where `place = 1<<c`.

Bitvector Representation



Placing a Queen in column `c` is not safe if

```
left & place != 0
```

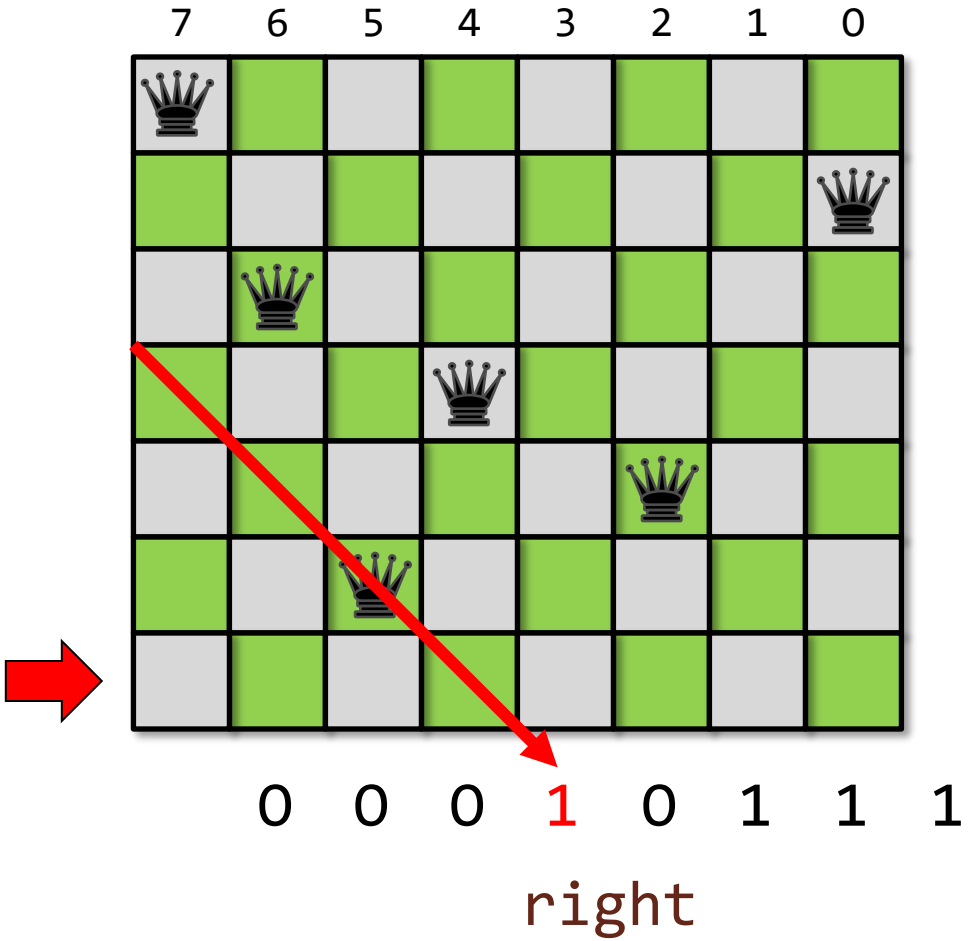
where `place = 1<<c`.

If column `c` is safe, then update

```
left = (left|place)<<1
```

for the next row.

Bitvector Representation

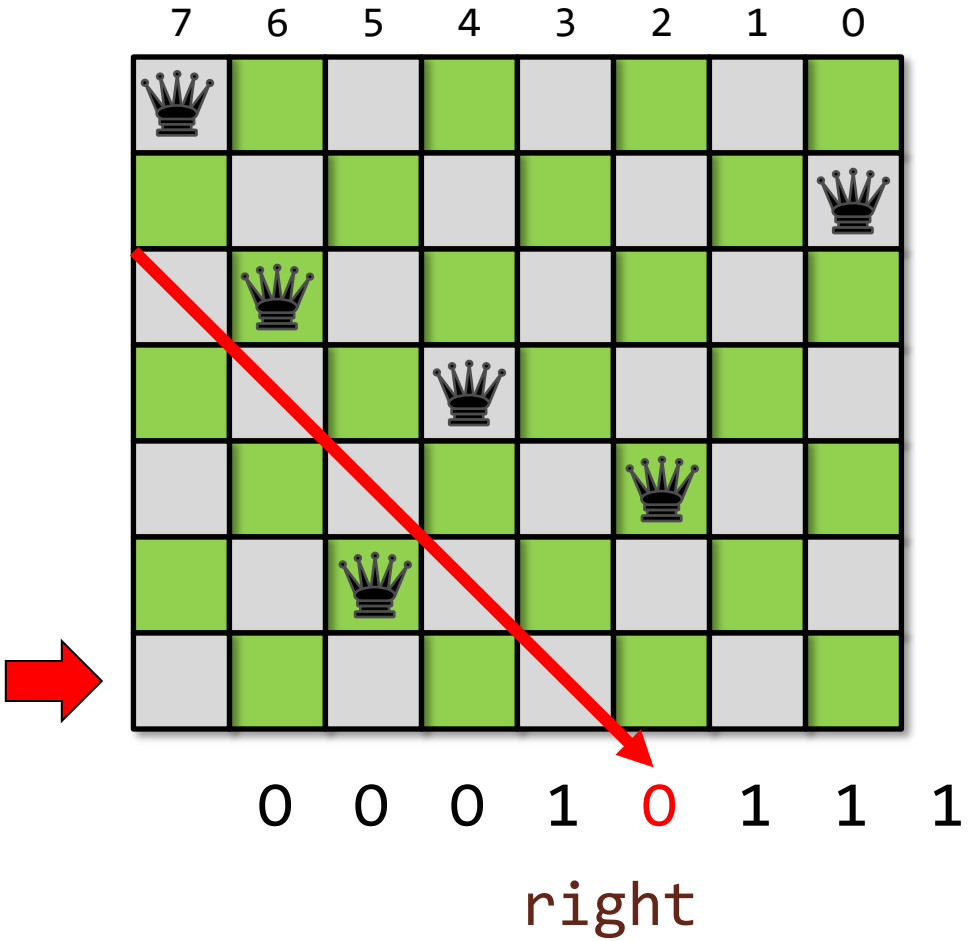


Placing a Queen in column `c` is not safe if

`right & place != 0`

where `place = 1<<c`.

Bitvector Representation

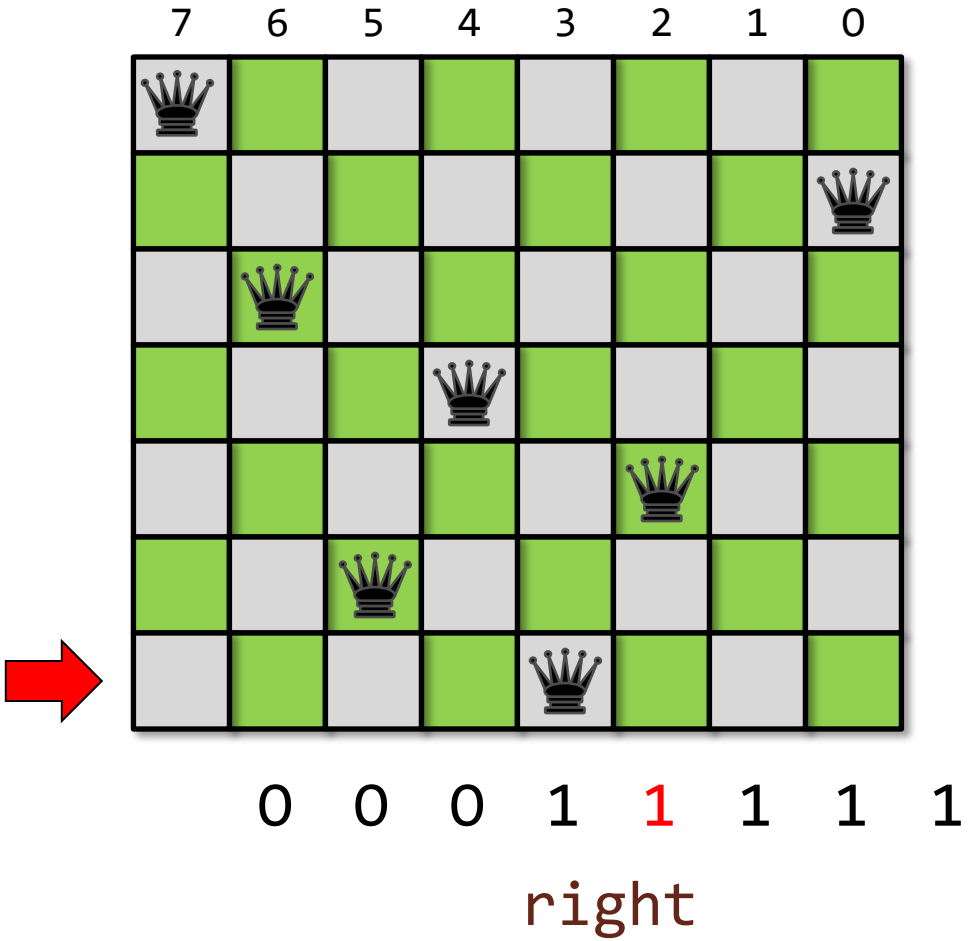


Placing a Queen in column `c` is not safe if

`right & place != 0`

where `place = 1<<c`.

Bitvector Representation



Placing a Queen in column `c` is not safe if

`right & place != 0`

where `place = 1<<c`.

If column `c` is safe, then update

`right = (right|place)>>1`

for the next row.

Queens Code

```
int32_t
queens(uint32_t mask,
       uint32_t down,
       uint32_t left,
       uint32_t right) {
  int32_t possible, place;
  int32_t count = 0;
  if (down == mask) return 1;
  for (possible = ~(down|left|right) & mask;
       possible != 0;
       possible &= ~place) {
    place = possible & -possible;
    count += queens(mask,
                    down|place,
                    ((left|place) << 1) & mask,
                    (right|place) >> 1);
  }
  return count;
}
```

Root call

```
queens((1<<n)-1, 0, 0, 0));
```

Inspired by Tony Lezard (1991).