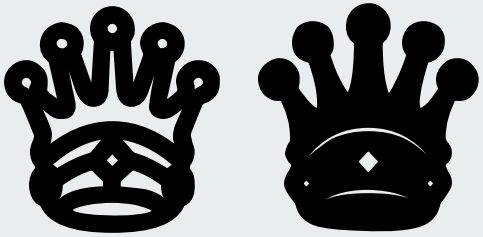
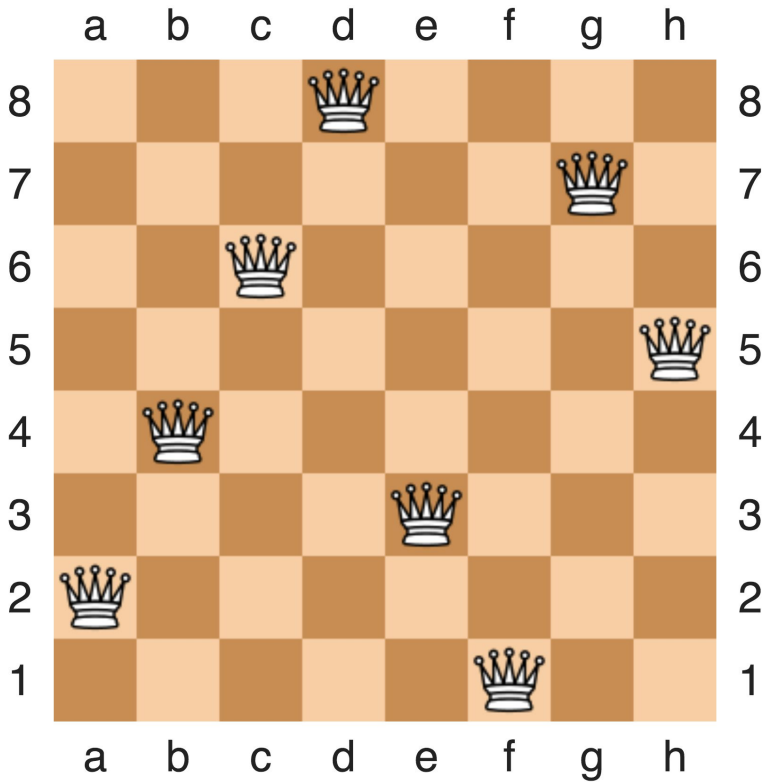
N-Queens



Simon Strassl – Michael Kainer – Johannes Vass

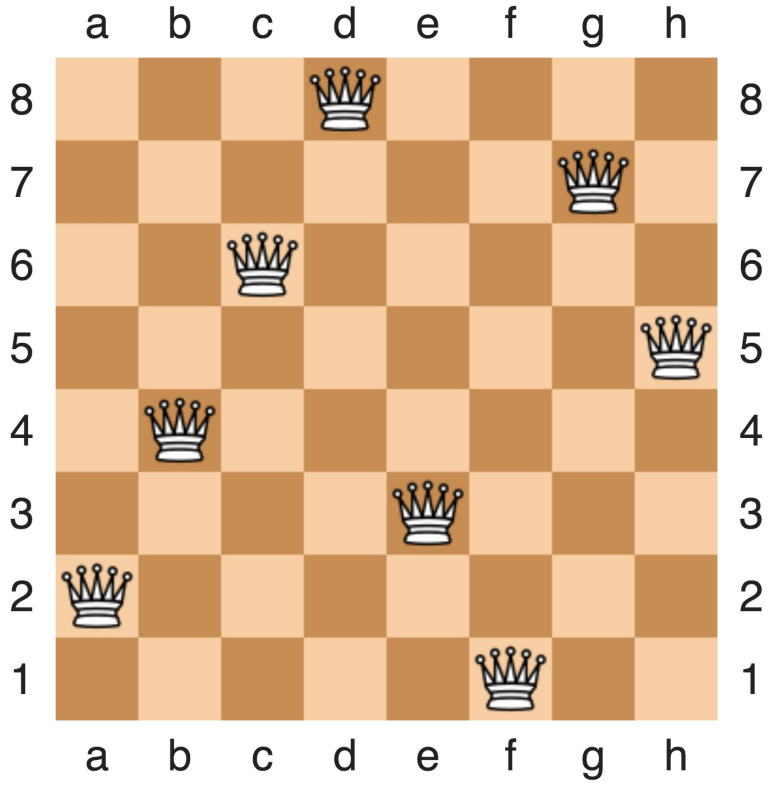
N-Queens Problem



https://en.wikipedia.org/wiki/Eight_queens_puzzle

Naive Lösung

- Methode: Generieren und Prüfen
- Rekursives Backtracking
- Board als bool Array
- Alle Damen gesetzt → Check aufrufen
- Laufzeit in O(choose(n^2, n))
- Testen mit bekannten Outputs



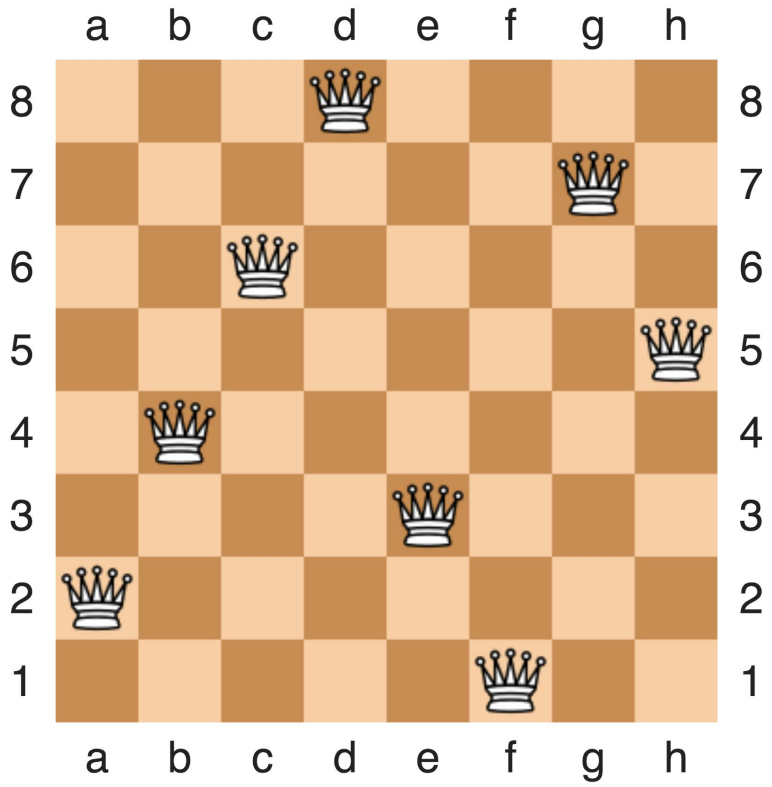
Filter eliminieren

× 8 Mio.

- Fixe Zeile pro Rekursionsebene
- Bedrohte Spalten mitschreiben
- Bedrohte Diagonalen mitschreiben

→ Nur noch valide Lösungen generieren

→ Kein Check mehr notwendig



Explizites Board entfernen

- Durch die vorige Optimierung wird es nicht mehr benötigt
- Jede vollständige Lösung ist valide

Vector<bool> -> bool[]

- Belegte Spalten und Diagonalen in std::vector<bool> markiert
- std::vector<bool> ist ein Bitvektor => teurer zu Indizieren

Iterieren über valide Spalten


```
for (int col = 0; col < n; ++col) {
  if (!used_columns[col]) { … }
}
```



```
for (int col_i = row; col_i < n; ++col_i) {
  int col = columns[col_i];
  std::swap(columns[row], columns[col_i]);
  …
  std::swap(columns[row], columns[col_i]);
}
```

bool[] -> uint32_t für Diagonalen

× 1.80

```
if (!used_diagonals_lb[lb_diagonal] &&
    !used_diagonals_rt[rt_diagonal]) {
  …
  used_diagonals_lb[lb_diagonal] = true;
  find_solutions(…, used_diagonals_lb, …)
  used_diagonals_lb[lb_diagonal] = false;
  …
}
```



```
if (((used_diagonals_lb & lb_diagonal) |
     (used_diagonals_rt & rt_diagonal)) == 0) {
  …
  find_solutions(…, used_diagonals_lb | lb_diagonal, …)
  …
}
```

Spalten Index Array -> uint32_t ✧


```
for (int col_i = row; col_i < n; ++col_i) {
  int col = columns[col_i];
  …
  std::swap(columns[row], columns[col_i]);
  find_solutions(…, columns, …)
  std::swap(columns[row], columns[col_i]);
  …
}
```

```
while (my_inverted_used_columns) {
  int col = __builtin_ctz(my_inverted_used_columns);
  my_inverted_used_columns &= my_inverted_used_columns - 1;
  …
  find_solutions(…, inverted_used_columns & ~(1 << col), …);
  …
}
```

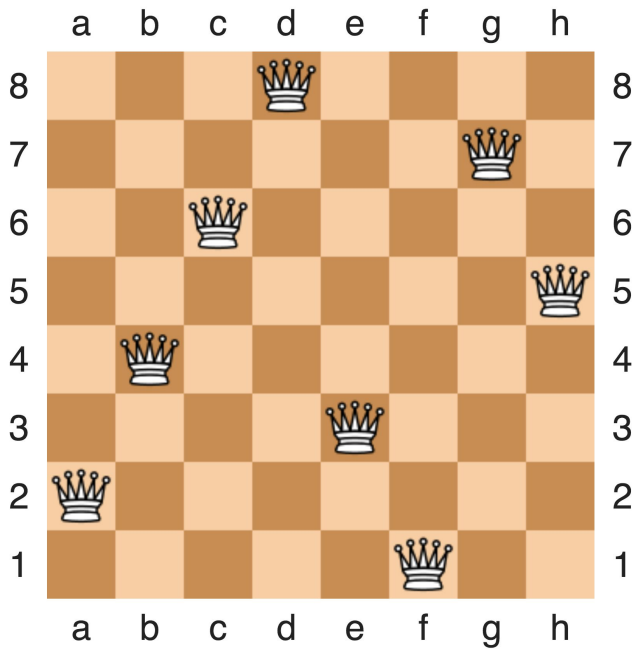
Optimiertes Bit-Clearing


```
static inline uint32_t clear_bit(uint32_t value, int bit) {
     asm("btr %1,%0" : "+r"(value) : "r"(bit), "r"(value) : "cc");
     return value;
}
```

Implizite Diagonalen

× 2.29

```
uint32_t my_inv_cols = inv_cols & ~diags_lb & ~diags_rt;
while (inv_cols) {
  int col = __builtin_ctz(my_inv_cols);
  uint32_t next_diags_lb = (diags_lb | (1 << col)) >> 1;
  uint32_t next_diags_rt = (diags_rt | (1 << col)) << 1;
  my_inv_cols ^= (1 << col);
  …
}
```



Mikrooptimierungen

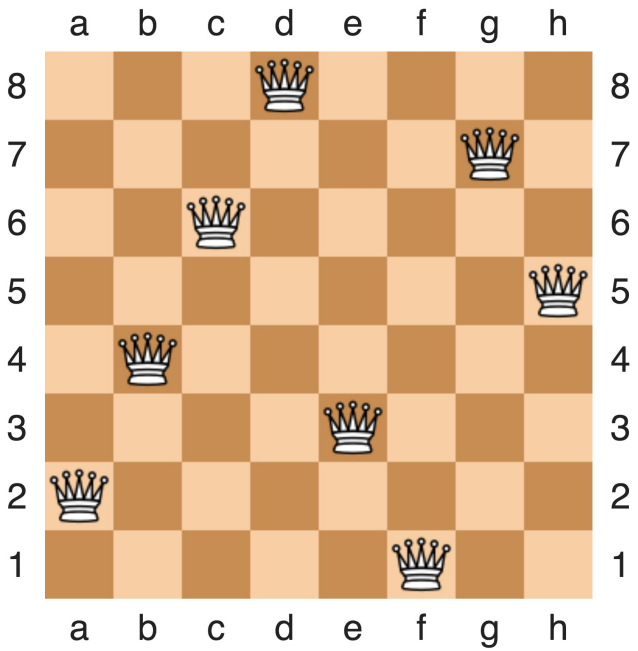
- Base Case in Schleife prüfen und solutions=1 setzen
- Invertierung der Bitfields entfernen
- n nicht mehr übergeben

Symmetrie

× 1.97

```
if constexpr (Stage == stage::first_even ||
              Stage == stage::first_odd ||
              Stage == stage::second_odd_center) {
    inverted_my_used_columns &= ~((1 << (n/2)) - 1);
}
```

```
if (n % 2 == 0) {
  return find_solutions<stage::first_even, 1>(...);
} else {
  return find_solutions<stage::first_odd, 1>(n,
used_columns, 0, 0);
}
```




x 188 Mio.