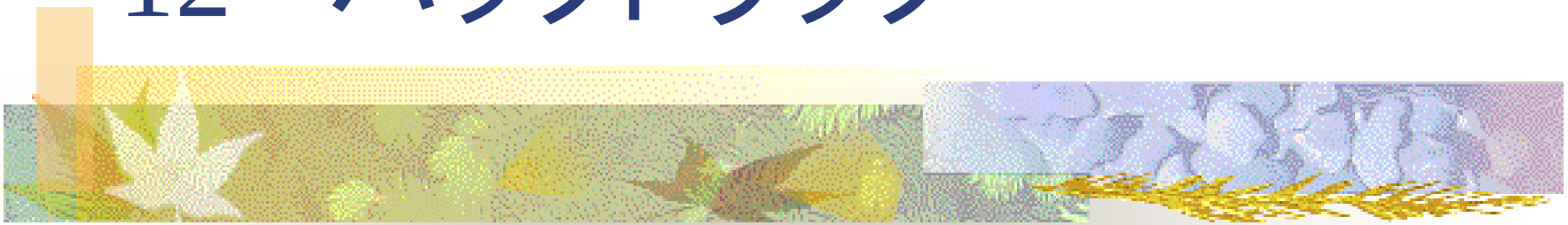
アルゴリズムとデータ構造b
12 - バックトラック



大見　嘉弘

バックトラック(後戻り法)

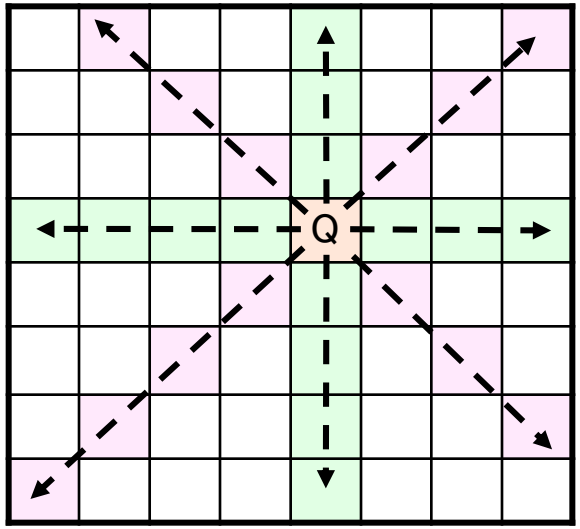
- それ以上先に進んでも解がない(やっても無駄な)場合、元来た道を戻って、違う道を辿る方法
- 探索する組み合わせが膨大な場合に、探索回数を大幅に減らせる
  - 探索の枝を切ってしまうことから「刈り込み」と呼ばれる。
  - チェス、将棋、囲碁などの解法、人工知能分野(推論、解決木の探索など)に有効、というか必須。
    (DeepBlueも刈り込みせずには人間に勝てない)

nクイーン問題

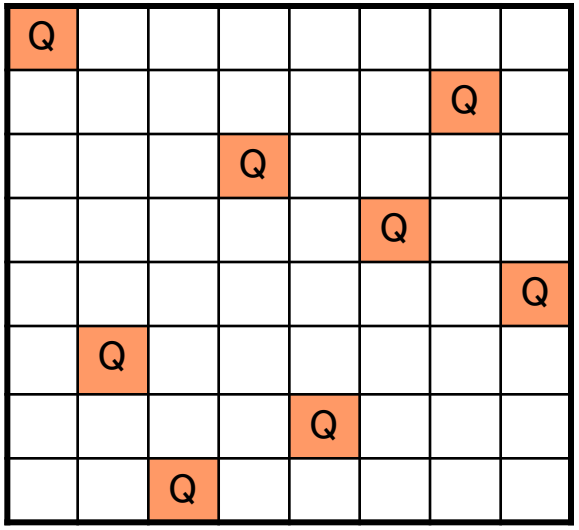
- バックトラックの例として取り上げる
- 最も一般的には8クイーン問題がある
  - ８×８の盤に8つのチェスのクイーンを置く
  - それぞれのクイーンは互いに取れない場所に置く

クイーンが移動できる範囲：
上下と斜めならいくつでも進める

将棋の角＋飛車



解の一例

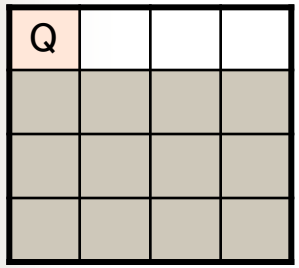


8クイーン問題での効果

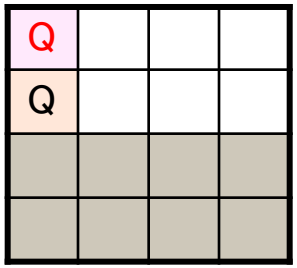
- 8クイーン問題の解は92個ある。
- クイーンを置く組み合わせは全部で
  $8^8 \fallingdotseq 1677$万通り
- それ以上進んでも解がない状態で、バックトラックを行うことで 15720通りだけ調べるだけで良くなる。
  - 1677万÷15720≒1067倍高速になる！

nクイーン問題の解法(1)

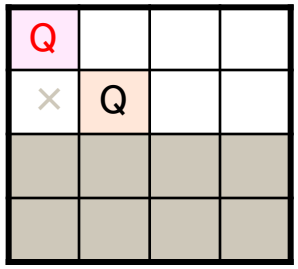
- 4クイーン問題
  - 各行に1つずつQ(クイーン)を置いてみる
  - Qは1列目、2列目…という順に置いてみる
  - 置けなかったら右の列へ、置けたら下の行に進む
  - 4行全部置けたら解が見つかった



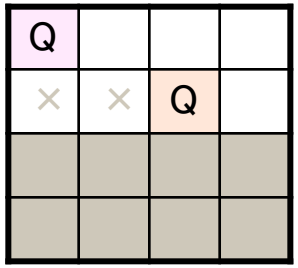
(1,1)に置ける
2行目に進む



(1,1)があるの
で(2,1)には置
けない

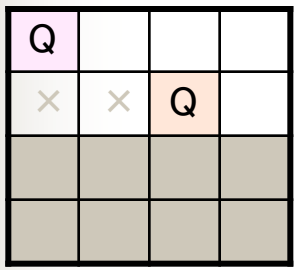


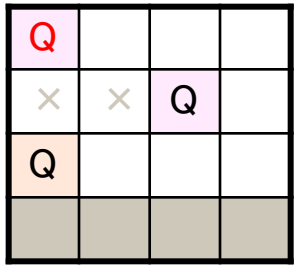
(1,1)があるの
で(2,2)には置
けない



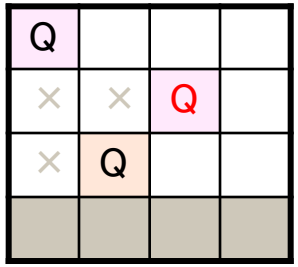
(2,3)に置ける
3行目に進む

nクイーン問題の解法(2)

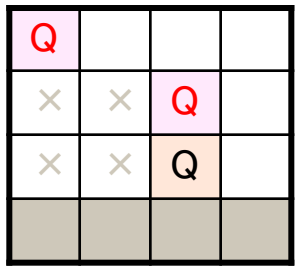




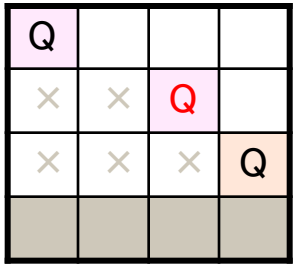
(1,1)があるので(3,1)には置けない



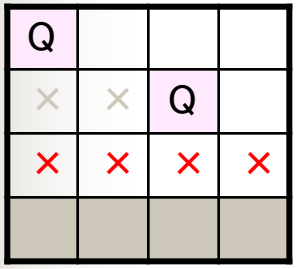
(2,3)があるので(3,2)には置けない



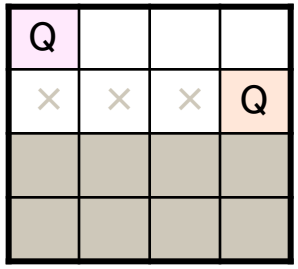
(1,1)と(2,3)があるので(3,3)には置けない



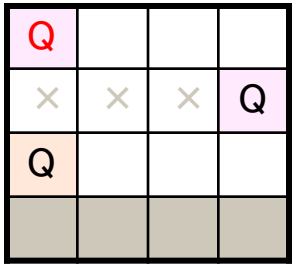
(2,3)があるので(3,4)には置けない



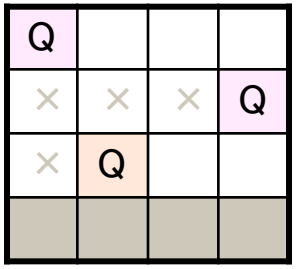
3行目は全てダメだったので、バックトラック(2行目に戻る)する



(2,4)に置ける
3行目に進む

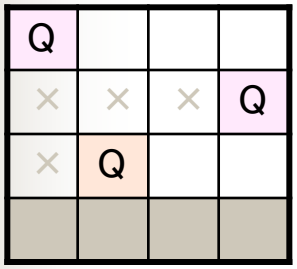


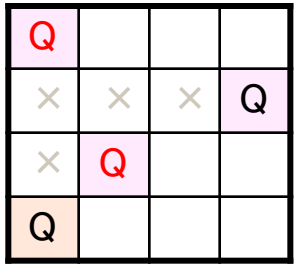
(1,1)があるので(3,1)には置けない



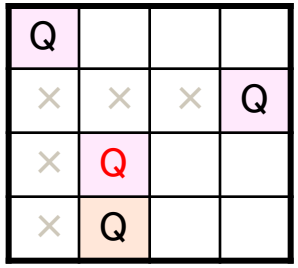
(3,2)に置ける
4行目に進む

nクイーン問題の解法(3)

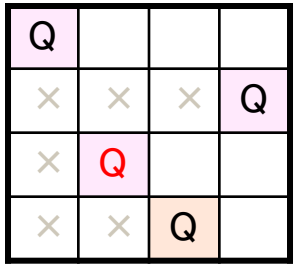




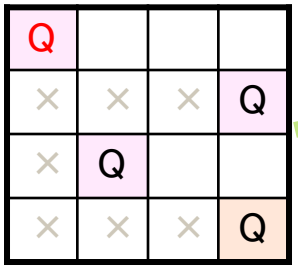
(1,1)と(3,2)があるので(4,1)には置けない



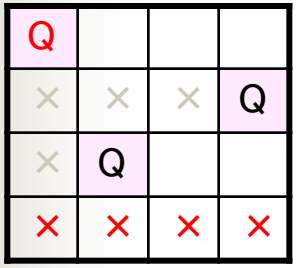
(3,2)があるので(4,2)には置けない



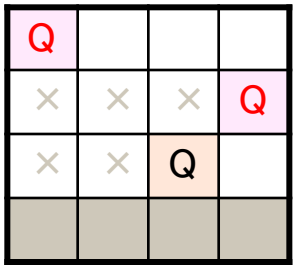
(3,2)があるので(4,3)には置けない



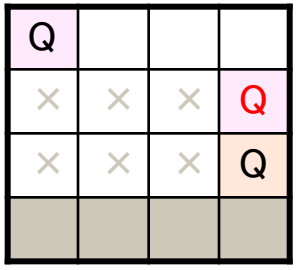
(1,1)があるので(4,4)には置けない



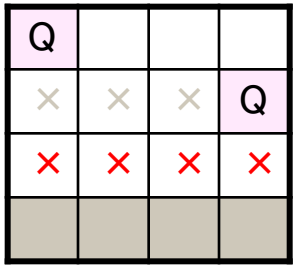
4行目は全てダメだったので、バックトラック(3行目に戻る)する



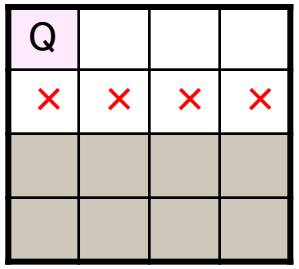
(1,1)と(2,4)があるので(3,3)には置けない



(2,4)があるので(3,4)には置けない

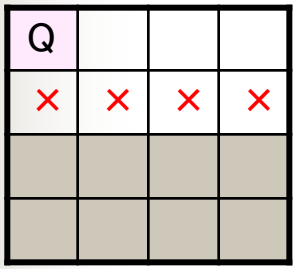


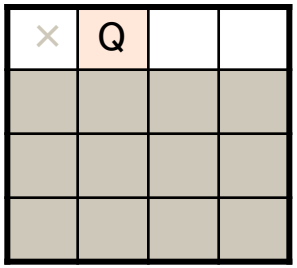
3行目は全てダメだったので、バックトラック(2行目に戻る)する



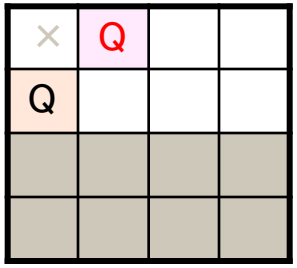
2行目は全てダメだったので、バックトラック(1行目に戻る)する

nクイーン問題の解法(4)

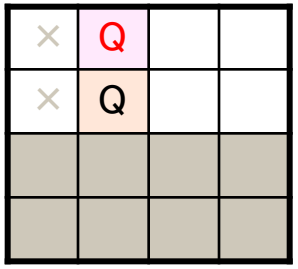




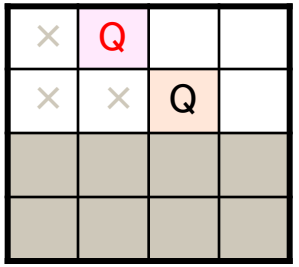
(1,2)に置ける
2行目に進む



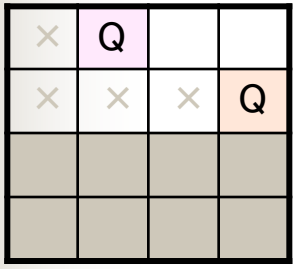
(1,2)があるので(2,1)には置けない



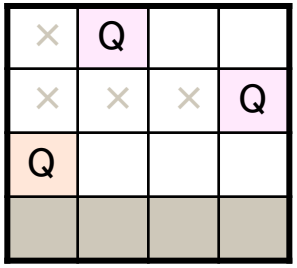
(1,2)があるので(2,2)には置けない



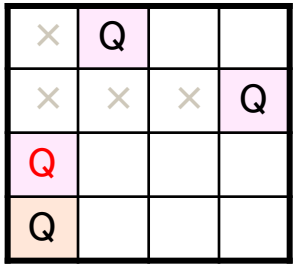
(1,2)があるので(2,3)には置けない



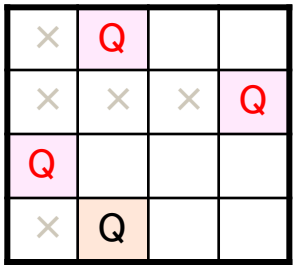
(2,4)に置ける
3行目に進む



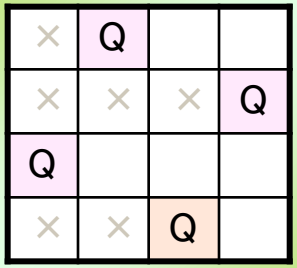
(3,1)に置ける
4行目に進む



(3,1)があるので(4,1)には置けない

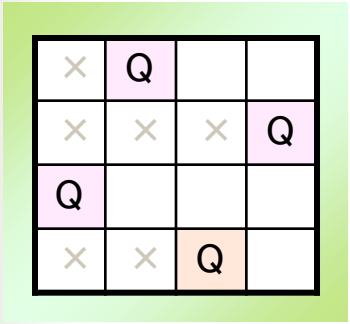


(1,2),(2,4),
(3,1)があるので(4,2)には置けない

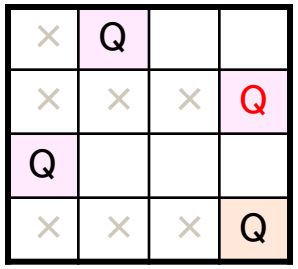


(4,3)に置ける
解発見！

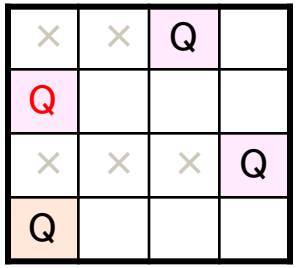
nクイーン問題の解法(5)



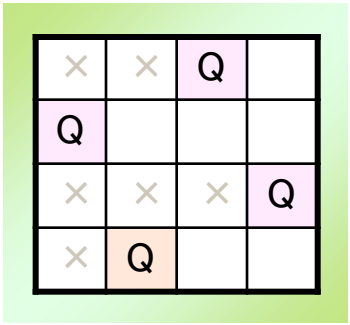
解発見の後は、
続き(4行目の
Qを一つ右へ)



(2,4)があるの
で(4,4)には置
けない

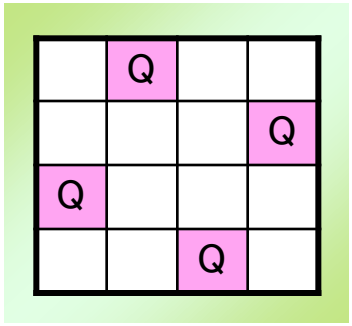


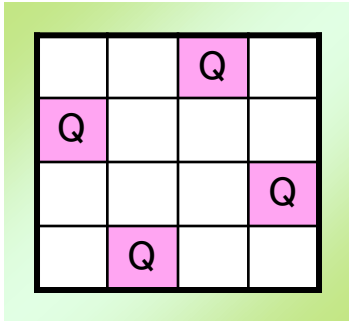
(2,1)があるの
で(4,1)には置
けない



(4,2)に置ける
解発見！

4クイーン問題の解は、
この2つだけ





プログラム例(変数や配列)

- 変数や配列

```
final int N=4;      // 盤の大きさ(N×N)
int[] line = new int[N+1];
                    // 各行のクイーンの位置
```

- lineの例

プログラム例(本体の概略)

```
public void solve(int i) { //i行目に置くQを求める
    int j,k;
    for (j=1; j<=N; j++) { //jはi行目にQを置く位置
        boolean b = true;
        // 同じ列にQがいるか？
        // 左上方向にQがいるか？
        // 右上方向にQがいるか？
        if (b) { // Qが置けるなら
            line[i]=j;
            if (i == N) { // 解発見！
                // 解を表示する
            } else {
                solve(i+1); //次の行にQを置く
            }
        }
    }
}
```

Qが取れる位置にいたらbをfalseに

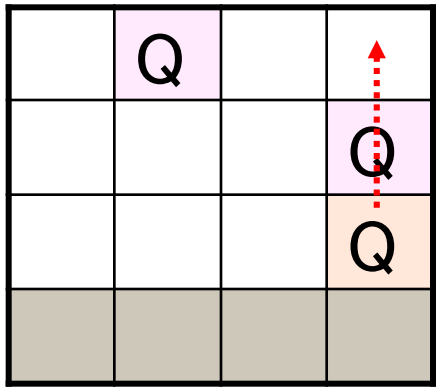
プログラム例(本体の前半)

```
public void solve(int i) { //i行目に置くQを求める
    int j,k;
    for (j=1; j<=N; j++) { //jはi行目にQを置く位置
        boolean b = true;
        // 同じ列にQがいるか?
        for (k=1; k<i; k++) {
            if (line[k] == j) b = false;
        }
        // 左上方向にQがいるか?
        for (k=1; k<i; k++) {
            if (line[k] == j-(i-k)) b = false;
        }
        // 右上方向にQがいるか?
        for (k=1; k<i; k++) {
            if (line[k] == j+(i-k)) b = false;
        }
        if (b) { // Qが置けるなら …
        }
    }
}
```

同じ列にQがいるか?

```
for (k=1; k<i; k++) {
    if (line[k] == j) b = false;
}
```

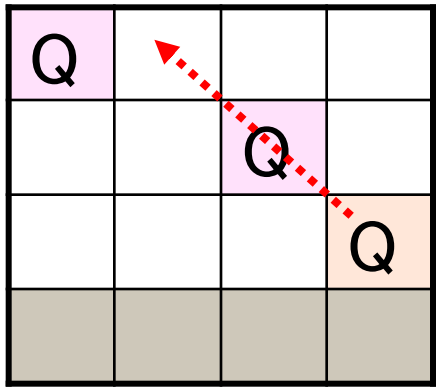
- 3行目(i=3)の4列目(j=4)にQを置く場合、
  - 1行目のQ(line[1])が4列目(j)にいないか? line[1]は2≠4
  - 2行目のQ(line[2]が4列目(j)にいないか?
    line[2]は4なので同じ列(4列目にQがいる)
    そこでbをfalseにする



左上にQがいるか?

```
for (k=1; k<i; k++) {
   if (line[k] == j-(i-k))
       b = false;
}
```

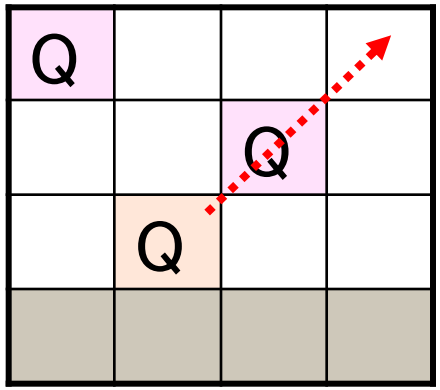
- 3行目(i=3)の4列目(j=4)にQを置く場合、
  - 1行目のQ(line[1])が2列目(j-2)にいないか? line[1]は1≠2
  - 2行目のQ(line[2]が3列目(j-1)にいないか?
    line[2]は3なので左上(3列目にQがいる)
    そこでbをfalseにする



右上にQがいるか?

```
for (k=1; k<i; k++) {
   if (line[k] == j+(i-k))
       b = false;
}
```

- 3行目(i=3)の2列目(j=2)にQを置く場合、
  - 1行目のQ(line[1])が4列目(j+2)にいないか? line[1]は1≠2
  - 2行目のQ(line[2]が3列目(j+1)にいないか?
    line[2]は3なので右上(3列目にQがいる)
    そこでbをfalseにする



Qが置ける場合…

i行目のj列目にQを置く

```
if (b) { // Qが置けるなら
   line[i] = j;
   if (i == N) { // 解発見!
      for(k=1; k<=N; k++) {
        System.out.print(line[k] + " ");
      }
      System.out.println();
   } else {
      solve(i+1); //次の行にQを置く
   }
}
```

解の表示

再帰

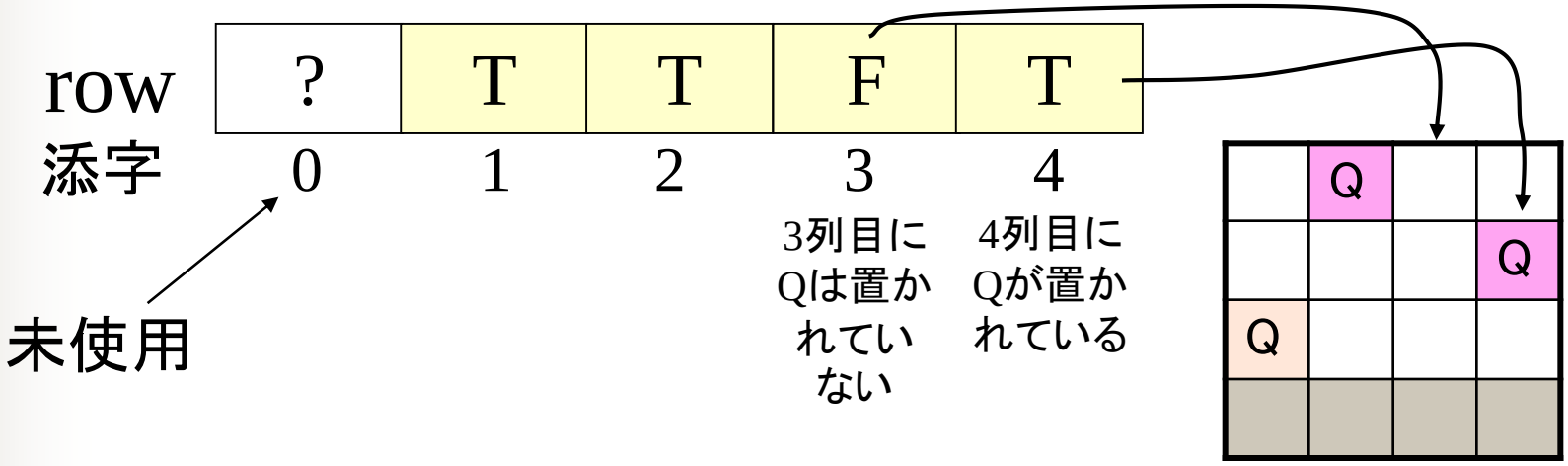
プログラムリスト(改良前)

```
public class NQueenSolver {
    final int N=4;
    int[] line = new int[N+1];
    public void solve(int i) { // i行目に置くQを求める
        int j,k;
        for (j=1; j<=N; j++) { // jはi行目にQを置く列番号
            boolean b = true;
             // 同じ列にQがいるか?
            for (k=1; k<i; k++) {
                if (line[k] == j) b = false;
            }
             // 左上方向にQがいるか?
            for (k=1; k<i; k++) {
                if (line[k] == j-(i-k)) b = false;
            }
             // 右上方向にQがいるか?
            for (k=1; k<i; k++) {
                if (line[k] == j+(i-k)) b = false;
            }
            if (b) { // Qが置けるなら
                line[i]=j;
                if (i == N) { // 解発見!
                    for(k=1; k<=N; k++) {
                        System.out.print(line[k] + " ");
                    }
                    System.out.println();
                } else {
                    solve(i+1); //次の行にQを置く
                }
            }
        }
    }
    public static void main(String[] args) {
        NQueenSolver prog = new NQueenSolver();
        prog.solve(1); //1行目から始める
    }
}
```

プログラムの改良(1)

- 同じ列、左上、右上にQがいないかどうか調べるのに、lineの各要素を見ている
- 一発でQがいないかどうか調べられないか?
- 同じ列：lineではなく各列のどこかにQが既に置かれているかどうかのフラグを使う

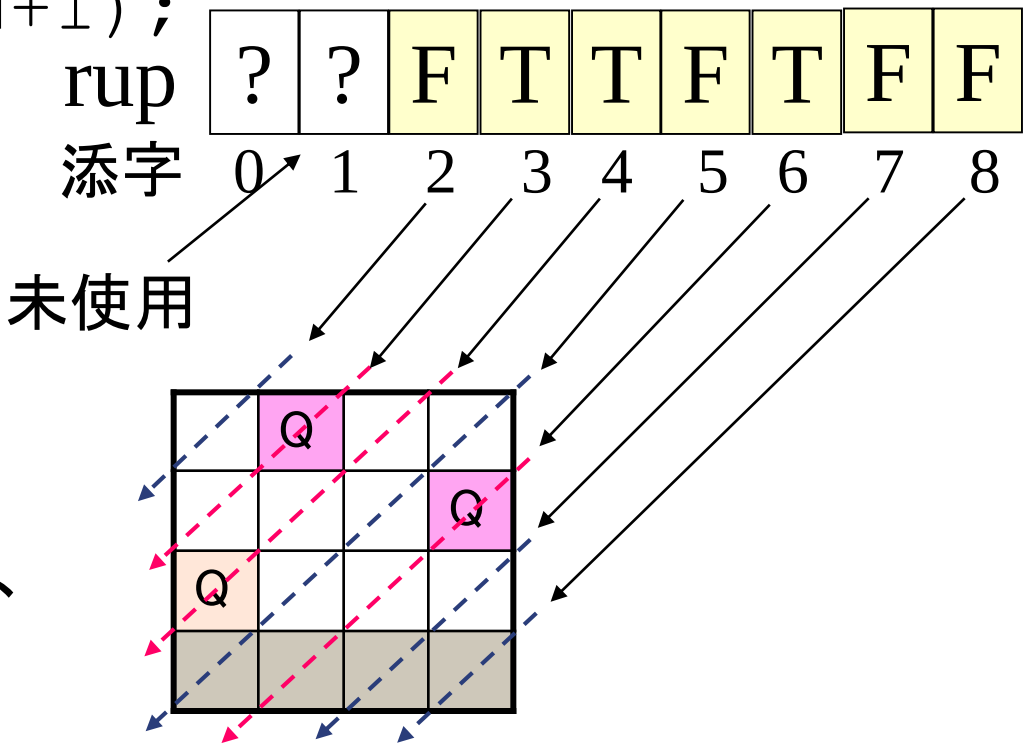
```
boolean row[] = new boolean(N+1);
```



プログラムの改良(2)

- 右上

```
boolean rup[] = new
boolean(2*N+1);
```



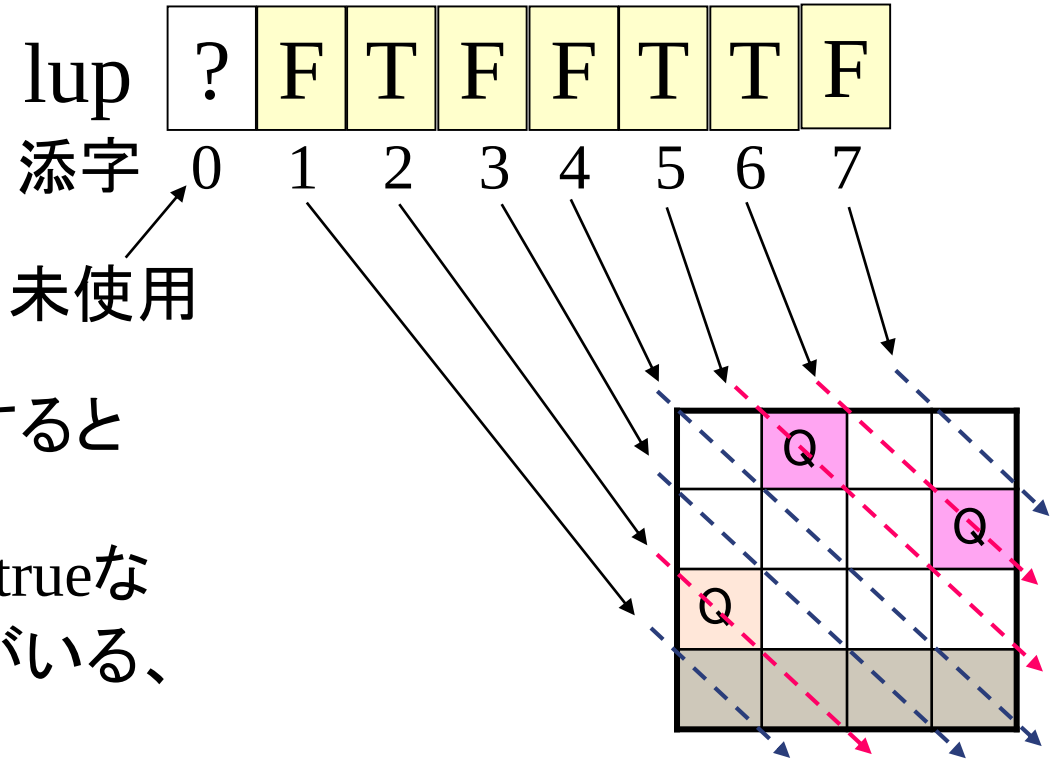
i行目,j列目とすると

rup[i+j]がtrueなら、
その筋にQがいる、
falseならいない

プログラムの改良(3)

- 左上

```
boolean lup[] = new boolean(2*N);
```



i行目,j列目とすると

rup[j-i+N]がtrueなら、その筋にQがいる、falseならいない

プログラムの改良(Qが置けるか?)

```
public void solve(int i) {
    int j,k;
    for (j=1; j<=N; j++) {
        if (row[j] == false &&
            rup[i+j] == false &&
            lup[j-i+N] == false) {
                pos[i]=j;
            …(省略)…
        }
    }
}
```

同じ列(j)にQがいないか?

同じ右上(i+j)にQがいないか?

同じ左上(j-i+N)にQがいないか?

プログラムの改良(次の行へ)

```
public void solve(int i) { // i行目に置くQを求める
    int j,k;
    for (j=1; j<=N; j++) { // jはi行目にQを置く列番号
        if (row[j] == false &&
            rup[i+j] == false &&
            lup[j-i+N] == false) {
            pos[i]=j;
            if (i == N) { // 解発見!
                // 解を表示
            } else {
                row[j] = rup[i+j] = lup[j-i+N] = true;
                solve(i+1); //次の行にQを置く
                row[j] = rup[i+j] = lup[j-i+N] = false;
            }
        }
    }
}
```

Qを置く

(j列目、(i+j)の右上斜め、(j-i+N)の左上斜めをtrueにする

Qを取り除く

同様の箇所をfalseにする

```
row[j] = rup[i+j] = lup[j-i+N] = true;
solve(i+1); //次の行にQを置く
row[j] = rup[i+j] = lup[j-i+N] = false;
```

プログラムリスト(改良後)

```
public class NQueenSolver2 {
    final int N=8;
    int[] pos = new int[N+1];
    boolean[] row = new boolean[N+1]; // 各列にQを置いたか?
    boolean[] rup = new boolean[2*N+1]; // 各右上斜めにQを置いたか?
    boolean[] lup = new boolean[2*N]; // 各左上斜めにQを置いたか?
    public void solve(int i) { // i行目に置くQを求める
        int j,k;
     for (j=1; j<=N; j++) { // jはi行目にQを置く列番号
            if (row[j] == false &&
                rup[i+j] == false &&
                lup[j-i+N] == false) {
                pos[i]=j;
                if (i == N) { // 解発見!
                    for(k=1; k<=N; k++) {
                        System.out.print(pos[k] + " ");
                    }
                    System.out.println();
                } else {
                    row[j] = rup[i+j] = lup[j-i+N] = true;
                    solve(i+1); //次の行にQを置く
                    row[j] = rup[i+j] = lup[j-i+N] = false;
                }
            }
        }
    }
    public static void main(String[] args) {
        NQueenSolver2 prog = new NQueenSolver2();
     prog.solve(1); //1行目から始める
    }
}
```

宿題

- 5クイーン問題を、この教材で挙げた方法で順に解を求め、1～4番目に見つかった解を答えよ
- 解答例: このような解を4つ書く