

Teamsの科目チームの情報



授業構成と成績評価

- 1講時は講義, 2講時は演習
- 講義資料の配布や、授業関係の連絡はすべてTeamsにて行う
- 講義資料は約1週間前に配布する(予習できるように)
- 演習課題は講義資料に含まれる
- 演習課題の提出: 課題を完成した時点でTAを呼び、確認してもらう
- 演習課題の提出期限: その都度知らせる
- 「確認テスト」を第8回(試験期間中の1/28)の授業で実施する
 - 教室のパソコンのみ使用可
- 成績評価: 演習50%, 確認テスト50%

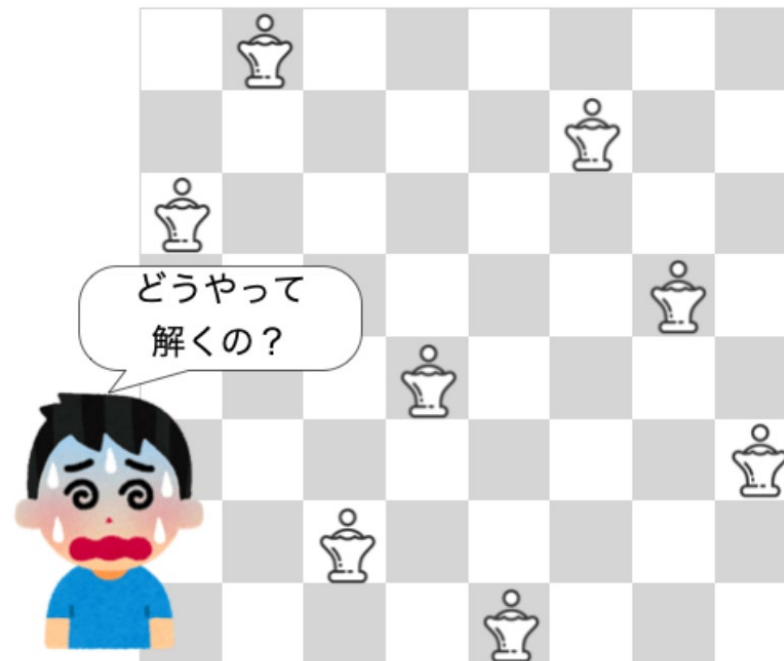
講義・演習

- 9:15までに教室のパソコンのWindowsにログインすること。9:20に自動出欠を取る
- 講義時は、Windowsパソコン上に映っている教員のパワーポイントを見ながら受講する。教室のスクリーンにも映す予定
- 演習時は、教室のパソコンをLinux(Ubuntu)で再起動して使用する。携帯PCもOK
 - 教室PCと携帯PC間のデータのやり取りはたとえばGoogle Driveを通じて行うことができる
- 演習時に質問等があれば挙手、TAさん・教員が直接対応
- 演習課題のチェックも挙手、TA・教員が直接対応

Nクイーン問題 (再帰アルゴリズム)

Nクイーン問題とは

- $N \times N$ のチェス盤上にクイーンを N 個置き、どのクイーンも他のクイーンに1手で取られないような配置にせよ、というパズル
- $N=8$ の場合は8クイーン問題(エイト・クイーン)

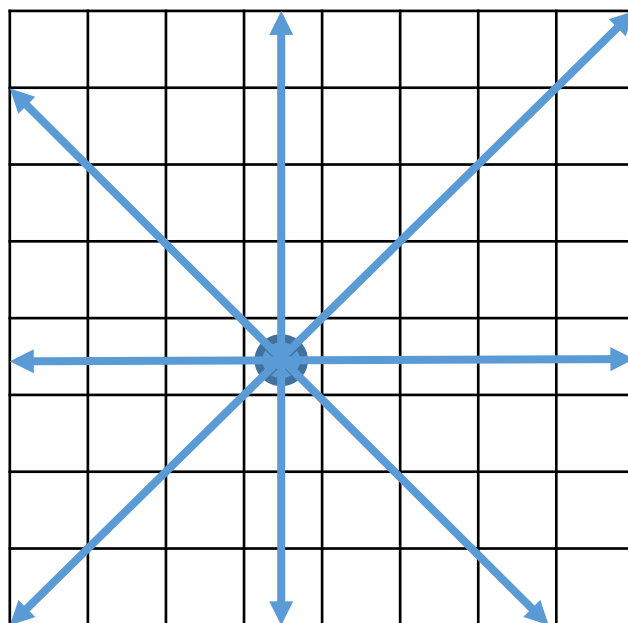


Nクイーン問題とは

- チェスのルールとして、クイーンは、図のように自分の場所から縦・横・斜めの方向に何マスでも動くことができる



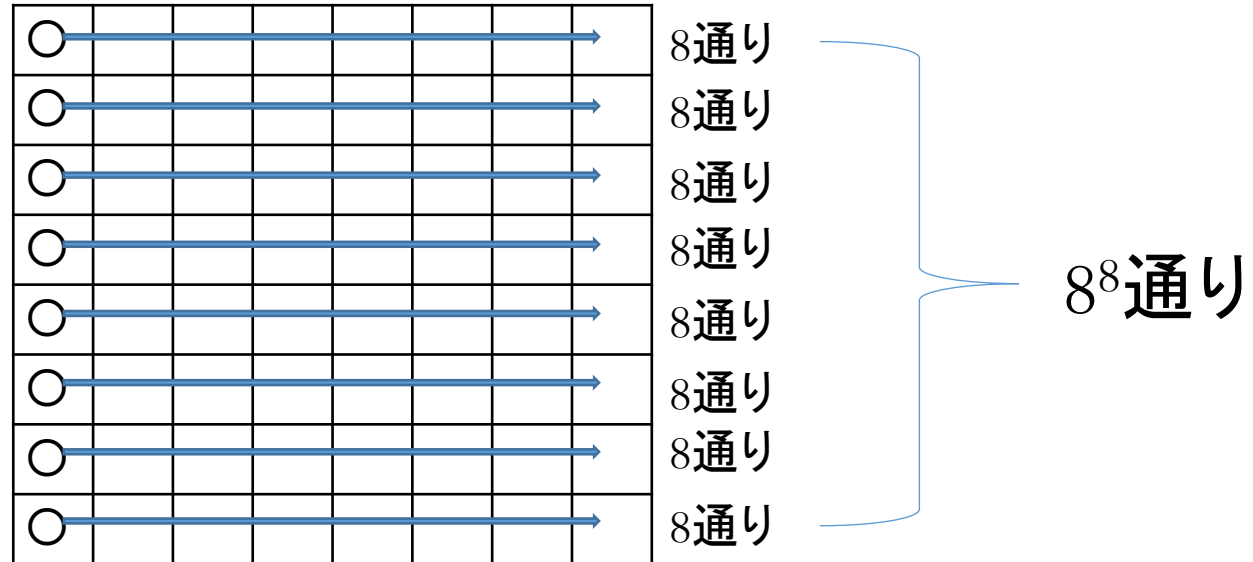
- コマ互いに横・縦・斜め上に配置しないようにしないといけない



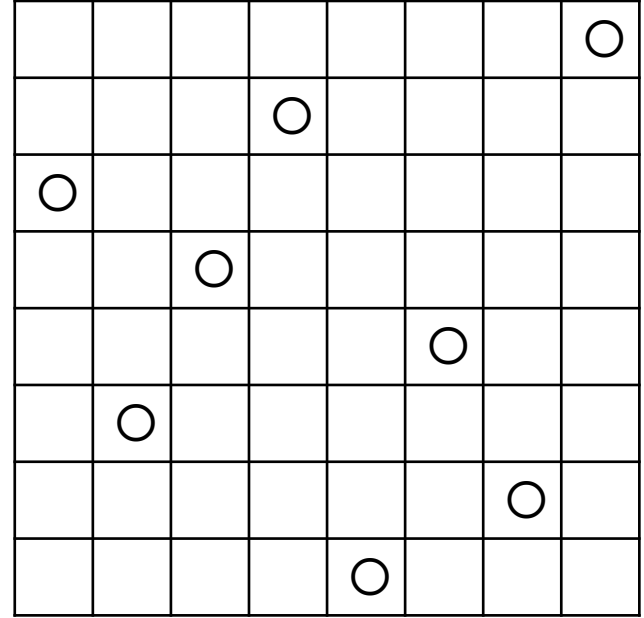
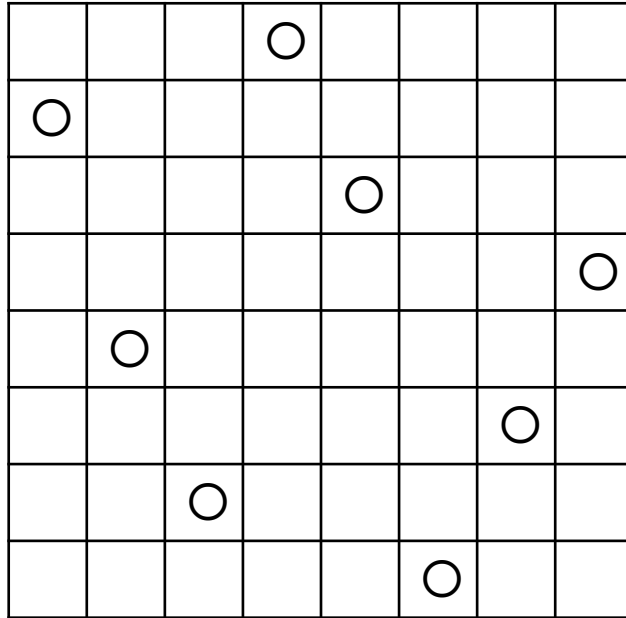
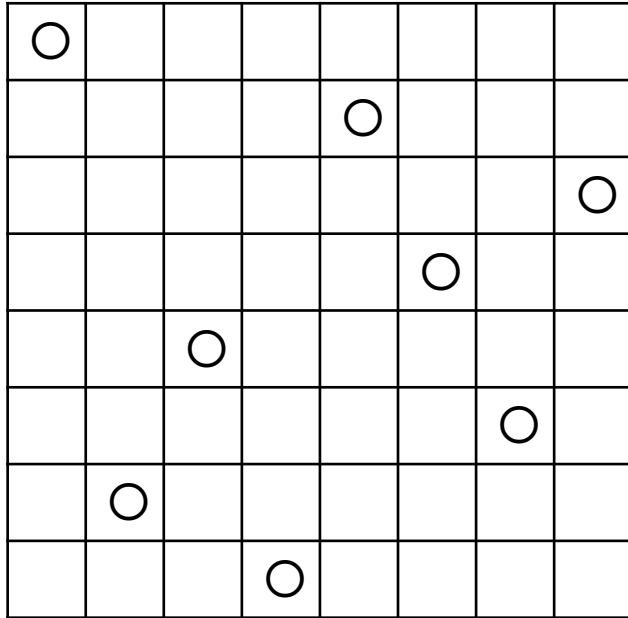
置き方の通り数

- $N=8$ の場合、 $C_{64}^8=4,426,165,368$ 通りの置き方...44億通り以上
- 「1行には1個のクイーンしか置けない」という制約を利用すれば、 $8^8=1,677,721,600$ の置き方...1千6百万以上

8⁸個の置き方(解候補)



解(計92個)の例



解き方

(「1行には1個のクイーンしか置けない」という制約を利用して)

- 方法1: すべての置き方を多重ループで作成・チェックする方法
- 方法2: 深さ優先探索で作成・チェックする方法(再帰)
- 方法3: 方法2へのちょっとした改良や、バックトラック法

多重ループの方法

- すべての場合 (解候補: 8^8) を8重のfor文で作成し、その一つ一つの置き方を調べる (解を求める) という方法

○							
○							
○							
○							
○							
○							
○							
○							

○							
○							
○							
○							
○							
○							
○							
	○						

...

○							
○							
○							
○							
○							
○							
○							
							○

○							
○							
○							
○							
○							
○							
	○						
○							

○							
○							
○							
○							
○							
○							
	○						
	○						

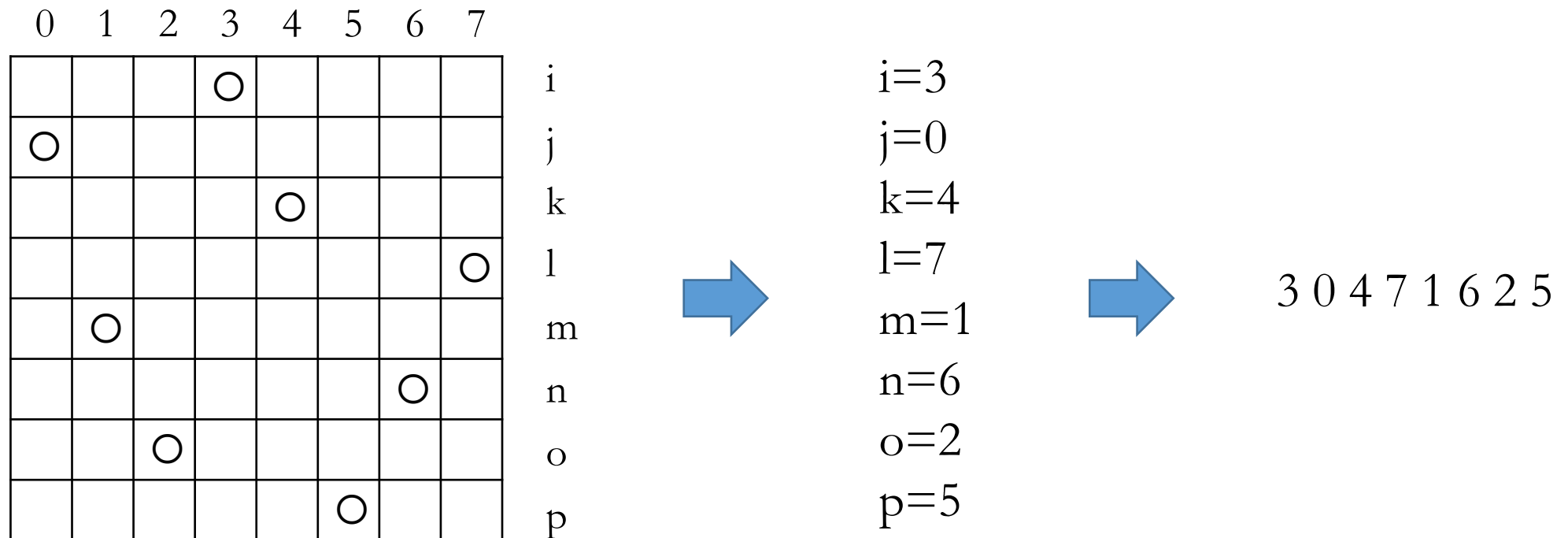
...

○							
○							
○							
○							
○							
○							
	○						
							○

⋮

解の表現

- 8×8 の盤面を左から右のマスに0から7の数字を与え、各行(横の列)にそれぞれ名前(変数名) i, j, k, l, m, n, o, p を付けるとすれば、その解(1つ)を右の各変数の値で表現できる



解の一部

0 4 7 5 2 6 1 3

0 5 7 2 6 3 1 4

0 6 3 5 7 1 4 2

0 6 4 7 1 3 5 2

1 3 5 7 2 0 6 4

1 4 6 0 2 7 5 3

1 4 6 3 0 7 5 2

1 5 0 6 3 7 2 4

1 5 7 2 0 3 6 4

解くポイント

- 変数 i, j, \dots, p を $0 \sim 7$ と変化させながらすべての場合を生成し、条件を満足する解を見つける
- すべての場合の生成は8重のfor文で実現できるはず
- 条件は以下のようなもの

1. 縦一直線上にあってはならない

$$i \neq j \neq \dots \neq p$$

2. 斜め直線上にあってはならない

- a. $|i - j| \neq 1, |j - k| \neq 1, \dots, |o - p| \neq 1$

- b. $|i - k| \neq 2, |j - l| \neq 2, \dots, |n - p| \neq 2$

⋮

- g. $|i - p| \neq 7$

なお、横一直線についての条件は
本手法の前提となっているので、
自然に満足している

	0	1	2	3	4	5	6	7	
				○					i
○									j
					○				k
								○	l
	○								m
							○		n
		○							o
						○			p

多重ループ方法の問題点

- N が大きくなると、プログラムの構造が非常に複雑になる
- そもそもfor文を使う以上、 N が固定のプログラムしか書けない



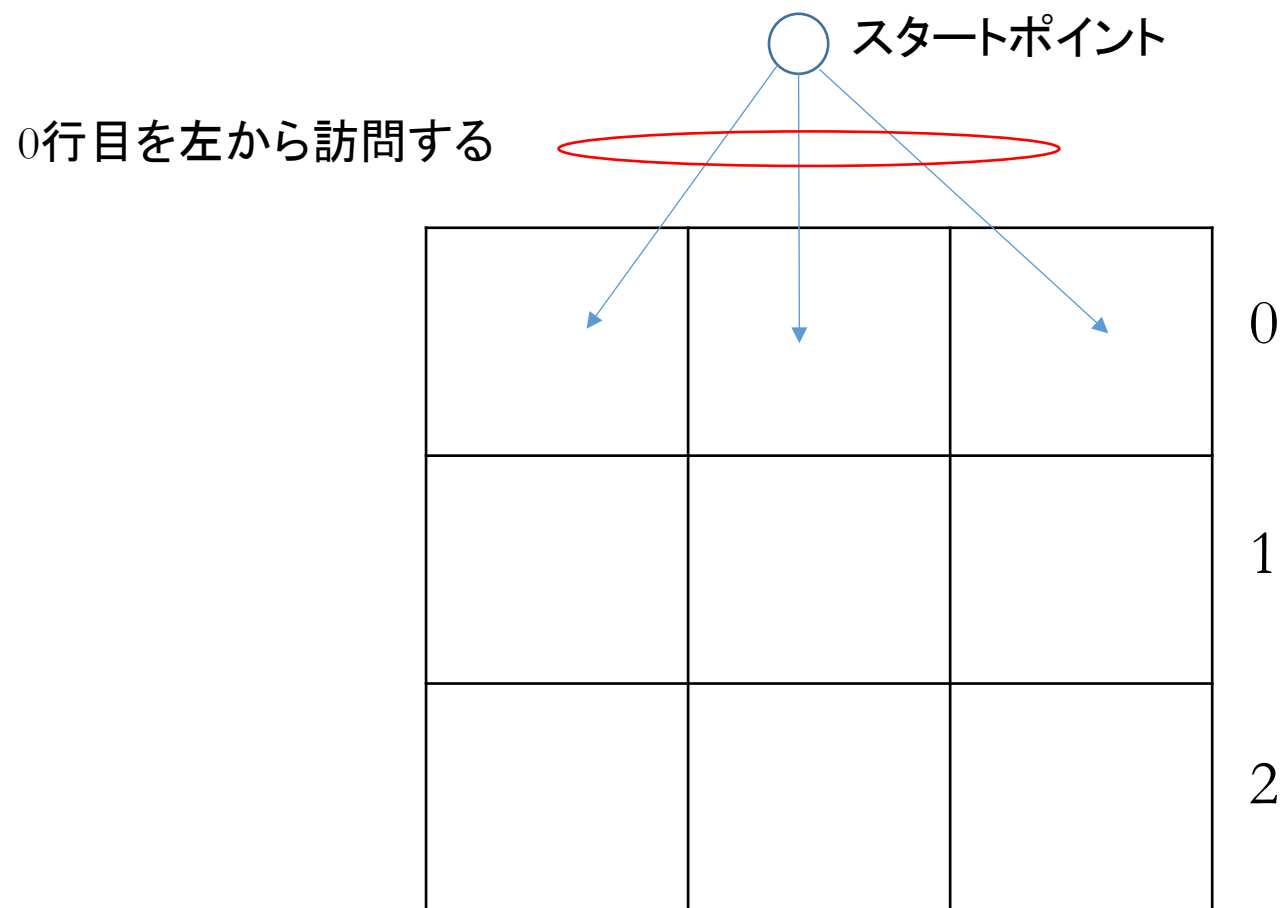
- 「深さ優先探索」というアプローチ（再帰で実現）

再帰とは

- 3Qで習った、「クイックソートの手順」の中に「クイックソートの手順」を呼び出す、というような、**自分自身を呼び出す**処理を再帰処理という。
- プログラム的に書くと、以下のようなもの。

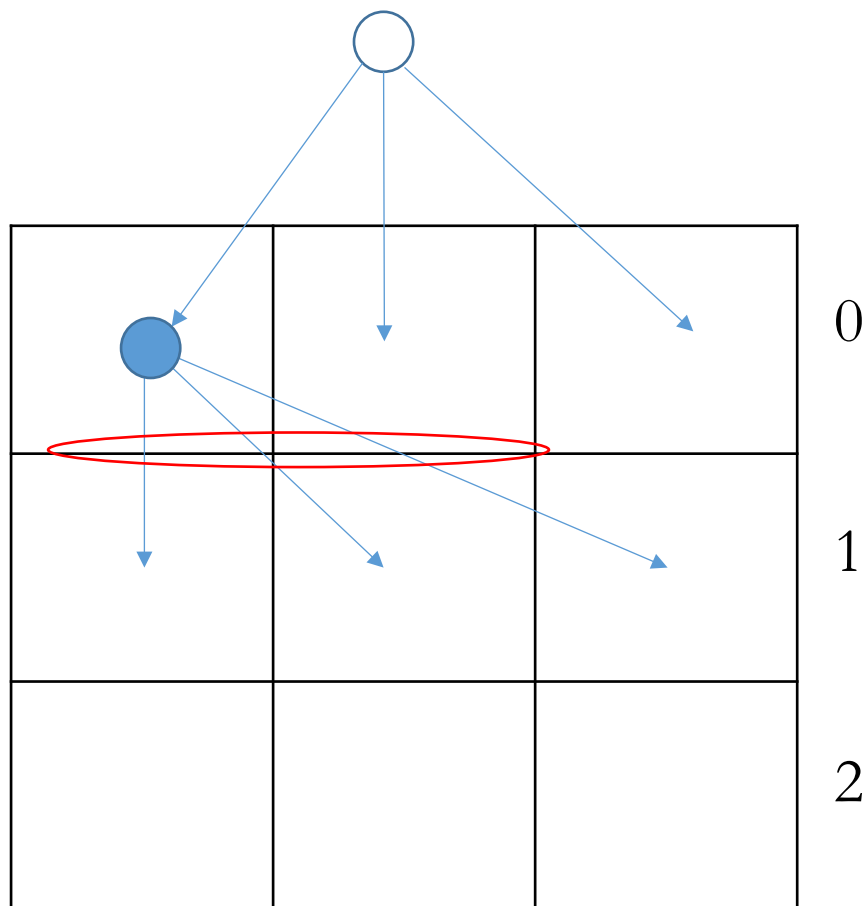
```
void proc() {  
    :  
    proc();  
    :  
}
```


Nクイーン問題の深さ優先探索



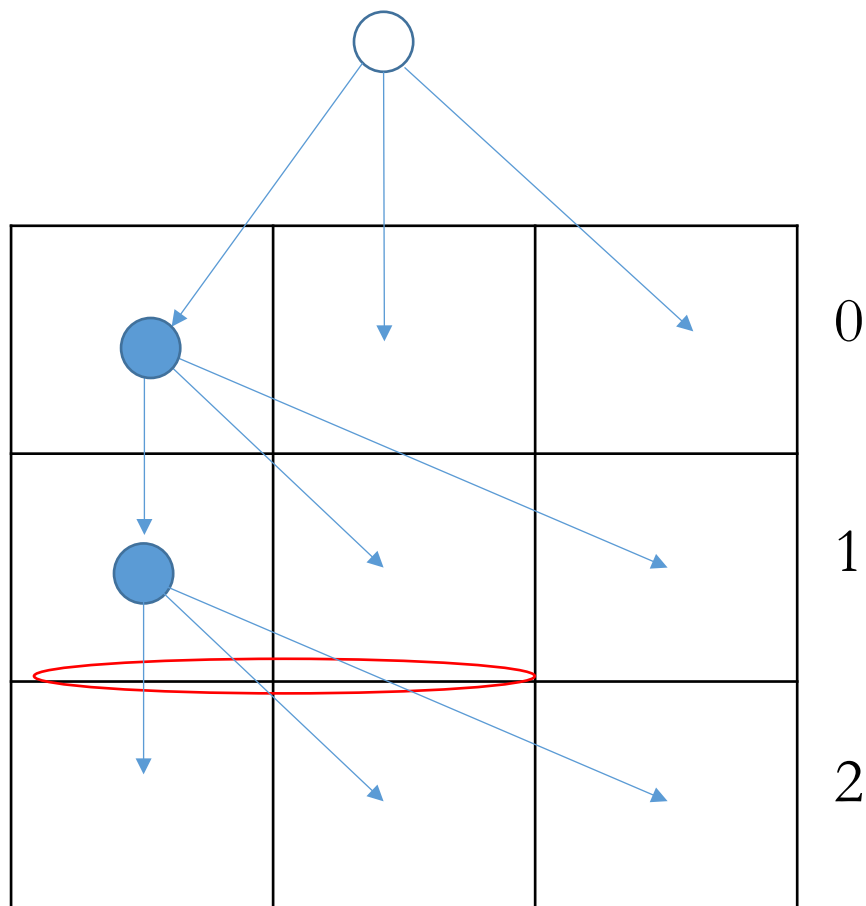
Nクイーン問題の深さ優先探索

0行目の一番左を訪問し、クイーンを置き、1行目を左から訪問する



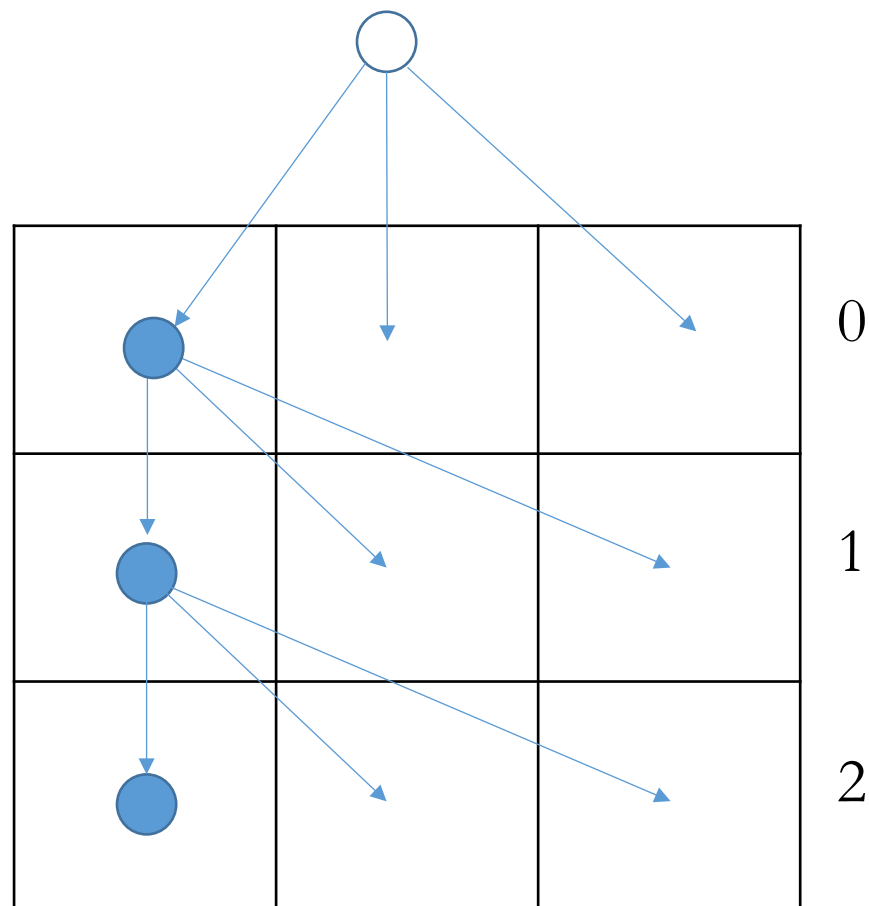
Nクイーン問題の深さ優先探索

1行目の一番左を訪問し、クイーンを置き、2行目を左から訪問



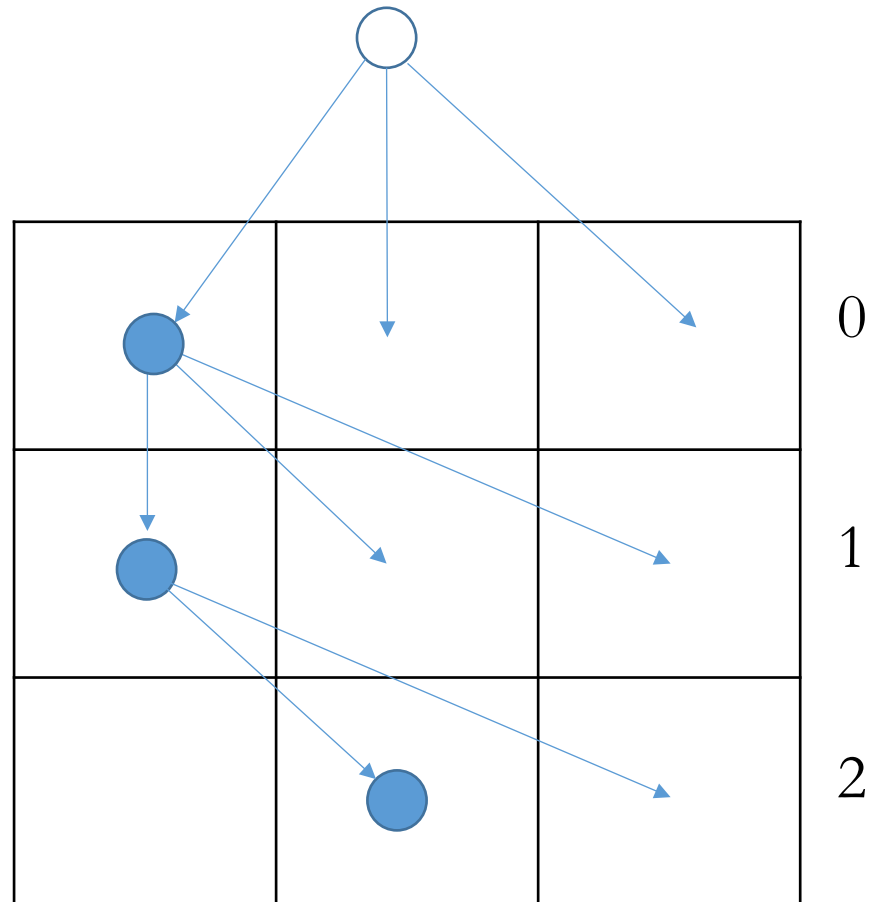
Nクイーン問題の深さ優先探索

2行目の一番左を訪問、クイーンを置き、(行き止まりなので)1つの解候補を得、同じ行のその次から訪問

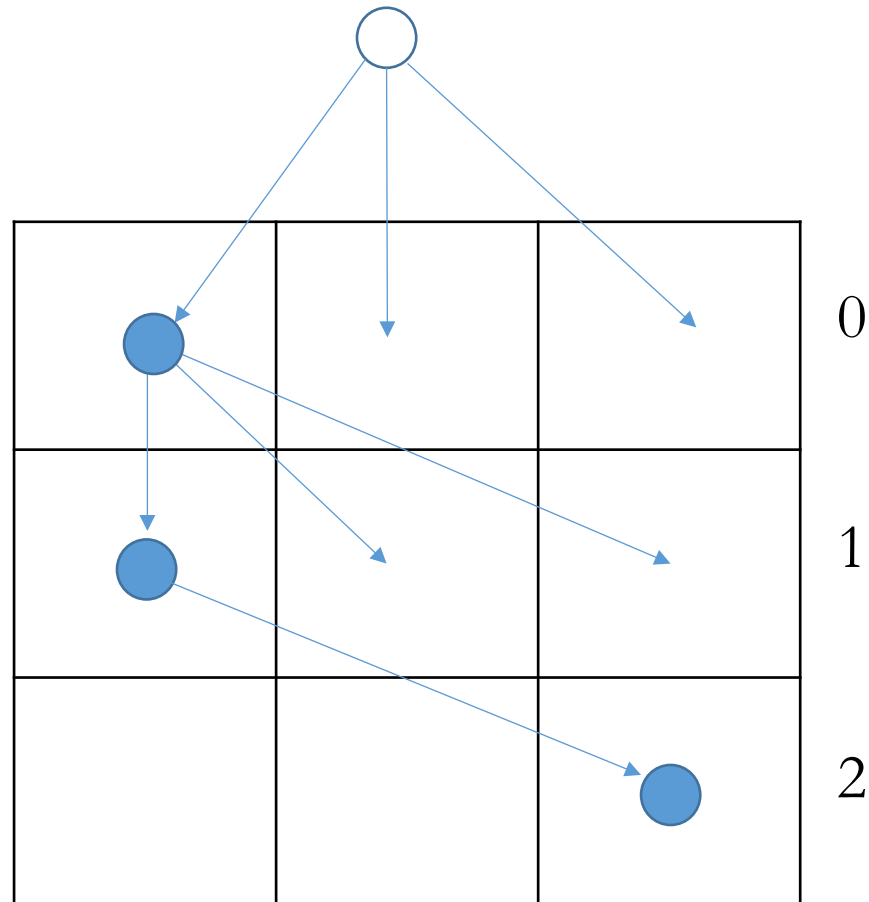


Nクイーン問題の深さ優先探索

中央を訪問しクイーンを置き、
(行き止まりなので) 1つの解候補
を得、同じ行の最も右を訪問



Nクイーン問題の深さ優先探索

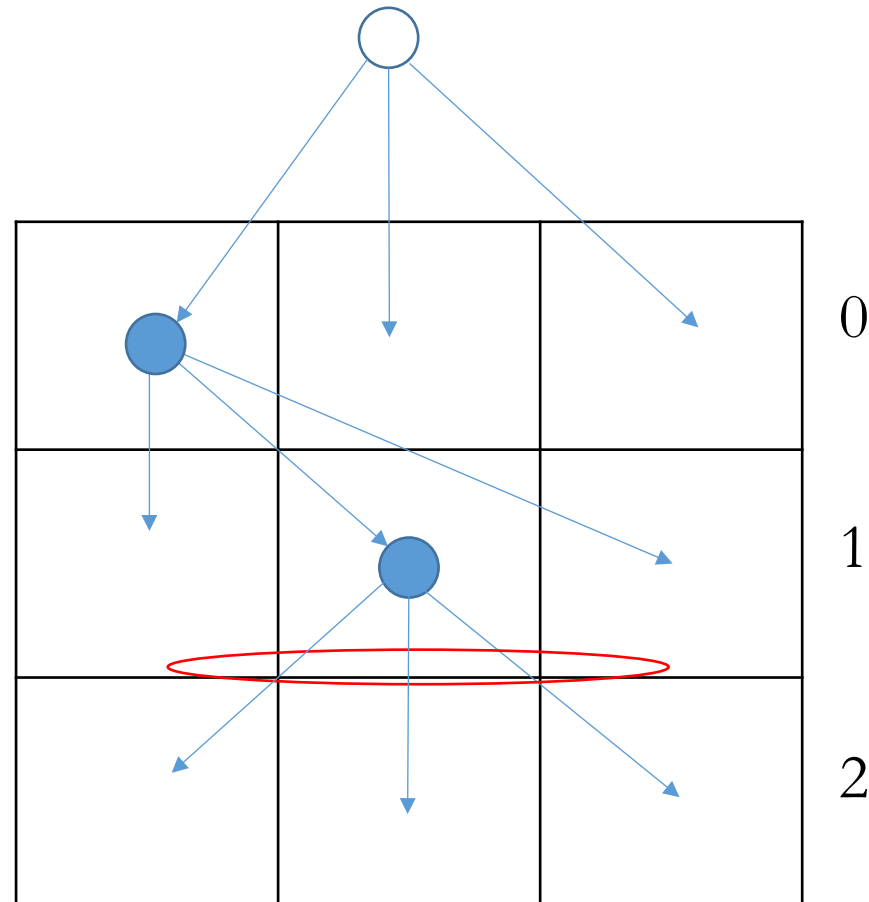


最も右を訪問しクイーンを置き、
(行き止まりなので)1つの解候補
を得る。この行が探索済みなので、
1行目に戻って未探索の場所から
訪問

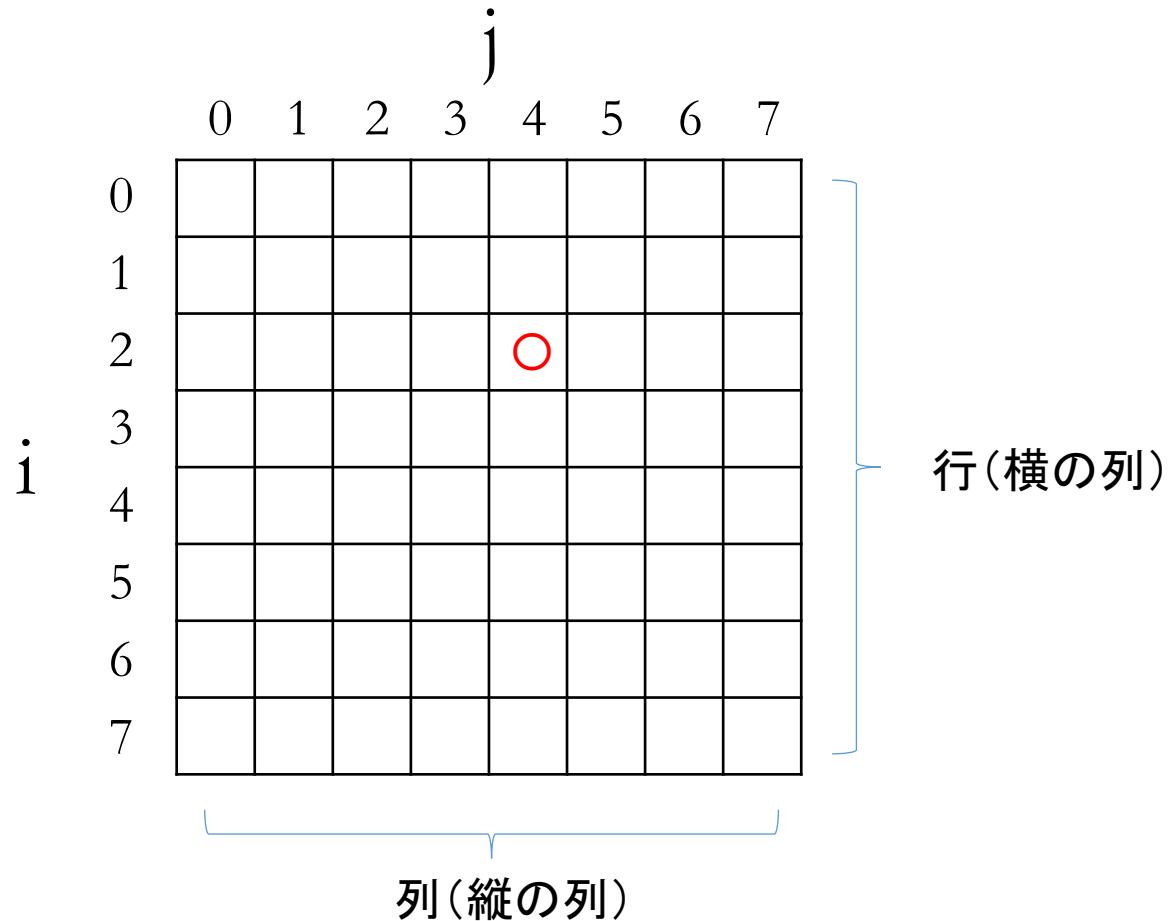
Nクイーン問題の深さ優先探索

1行目の中央を訪問しクイーンを置き、2行目を左から訪問。

このようにしていくと、0行目に戻る。ここからさらに探索していく。最終的にすべての場所が探索済みとなって、処理が終了



解の表現



1行にクイーン1個しかおけないので、
 i 行目の、クイーンの置かれた場所
(列番号)を、 $q[i]$ で記憶すればよい。

左図の場合： $q[2]=4$

つまり、 $q[i]$ が分かれば、

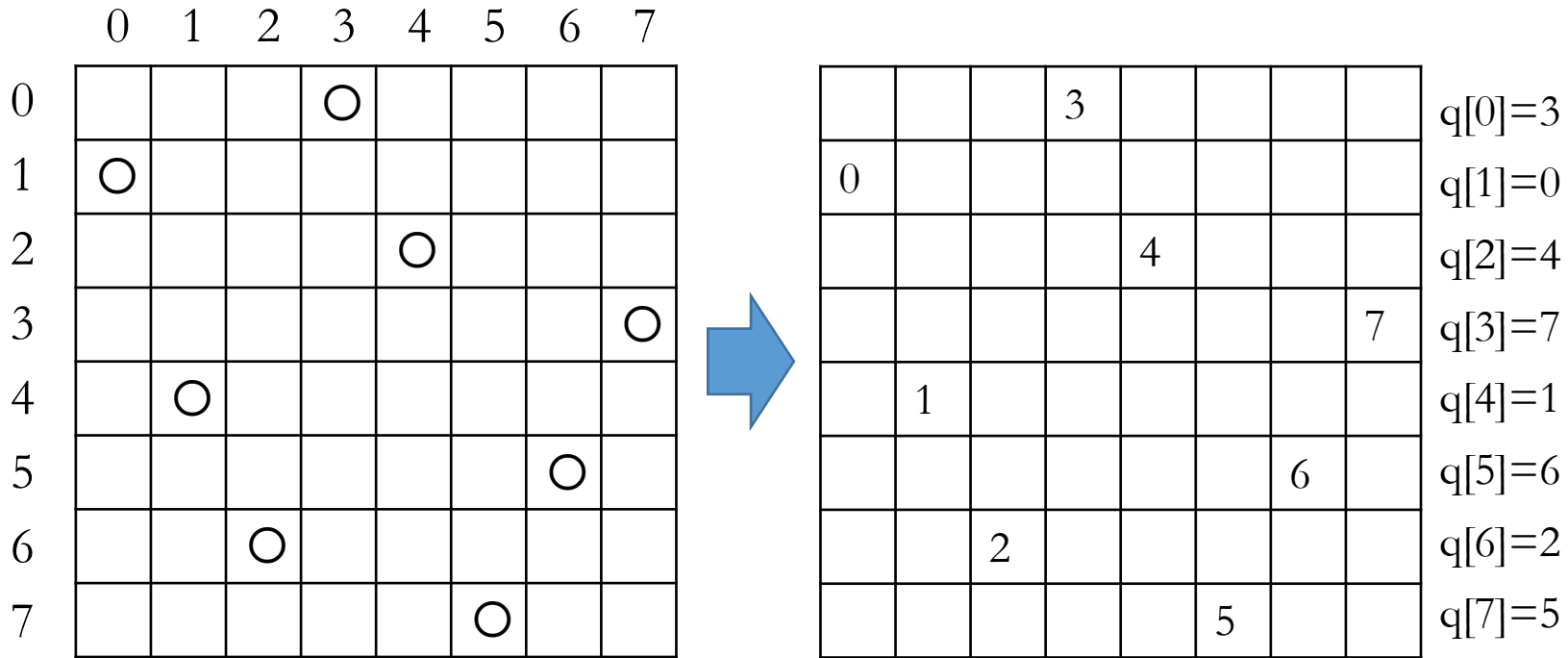
$(i, j)=(i, q[i])$... i 行目 $q[i]$ 列目

にクイーンが置かれていることがわかる

左図の場合 $i=2, j=q[2]=4$

$q[i]$ は1次元配列 q の i 番目の要素

解の表現



解

$q: 3\ 0\ 4\ 7\ 1\ 6\ 2\ 5$ or $(0,3)\ (1,0)\ (2,4)\ (3,7)\ (4,1)\ (5,6)\ (6,2)\ (7,5)$

Question

- 解として、8クイーンが以下のように配置された場合、その解を表現する1次元配列 q の中身を答えなさい。

	0	1	2	3	4	5	6	7
0								○
1			○					
2	○							
3						○		
4		○						
5					○			
6							○	
7				○				

Question

- $q[5]=3$ だとして、クイーンはどこに置く？

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

結局、Nクイーン問題は

- 左の探索空間から解候補 q (すべて)を深さ優先探索で見つけ、それらをチェックして解 q (すべて)を求めること

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

探索空間

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

解候補 q
解×

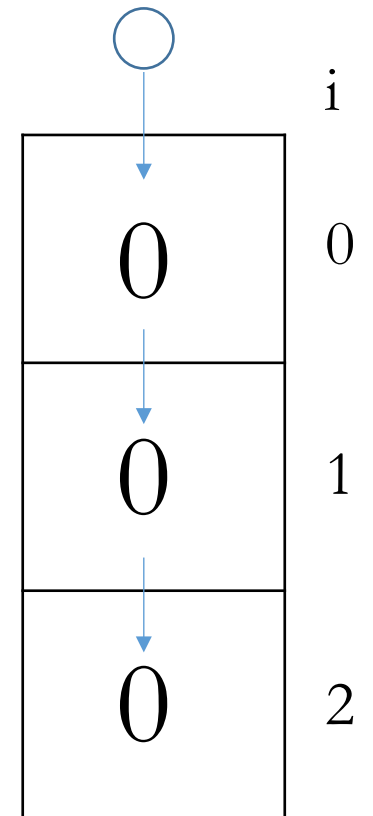
	0	1	2	3	4	5	6	7
0								7
1			2					
2	0							
3						5		
4		1						
5					4			
6							6	
7				3				

解候補 q
解○

盤面が縦1列のプログラム

- ここでは0列目を考える。つまり、
- $N=3$ なら1次元配列 q を、 $q[0]=0$, $q[1]=0$, $q[2]=0$ の順に再帰的にセットする
- そのため、
 - $i=0$ とし、 i 行目を訪問し、 $q[i]=0$ をセット
 - 次の行 ($i=1$) を訪問し、 $q[i]=0$ をセット
 - 次の行 ($i=2$) を訪問し、 $q[i]=0$ をセット
 - 訪問すべき行がないので終了

ということができればよい



盤面が縦1列のプログラム

- プログラムの基本形

```
void visit(int q[], int i) // iを訪問
{
    q[i]=0;
    visit(q, ?); // 次の行を訪問
}
int main(void)
{
    visit(q, 0); // i=0を訪問
    return 0;
}
```

盤面が縦1列のプログラム

- プログラムの基本形

```
void visit(int q[], int i)
{
    q[i]=0;
    visit(q, ?);
}
int main(void)
{
    visit(q, 0); // i=0を訪問
    return 0;
}
```



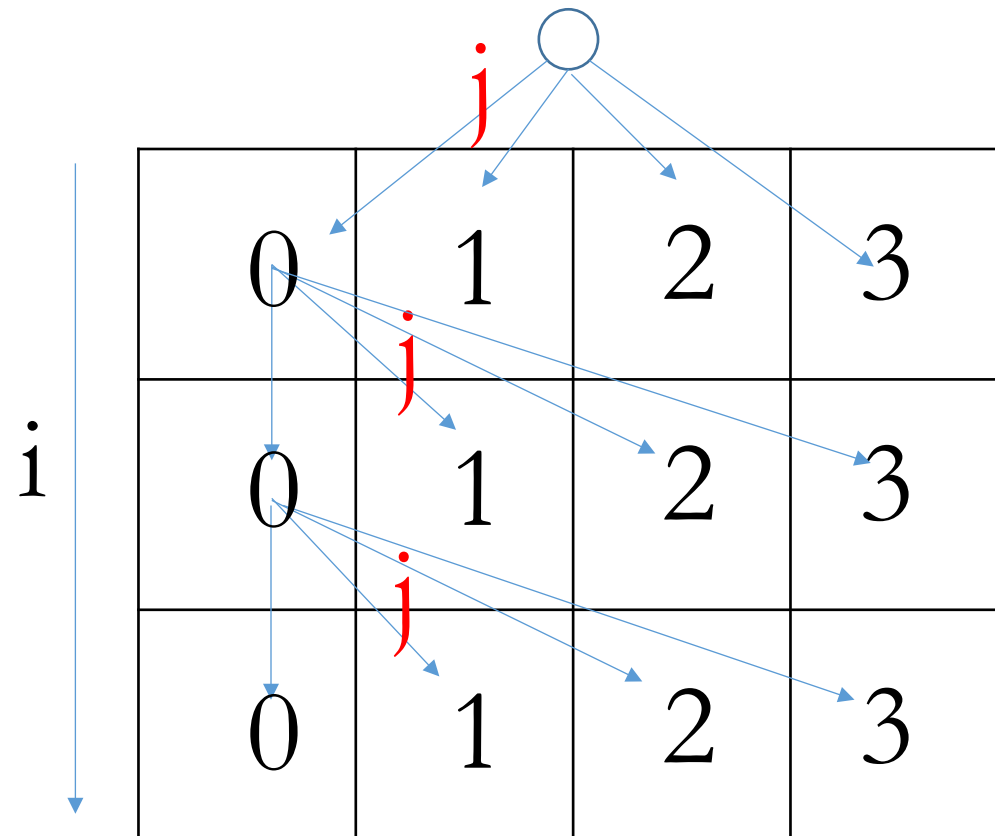
- しかしこれだと無限ループになってしま
う！！
- N個置かれたら終わるように「終了チェック」
を入れる必要がある



- iとNの関係をチェックすればよい

盤面が縦P列のプログラム

- i 行目の特定のマスから $i+1$ 行目を訪問するとき、 $i+1$ 行目の複数(すべて)のマスへの訪問を考慮しなければならない。
- そのため、`visit()`関数の中に列のためのループ文を加える必要がある



盤面が縦P列のプログラム

- プログラムの基本形 (赤い部分が縦1列と違うところ)

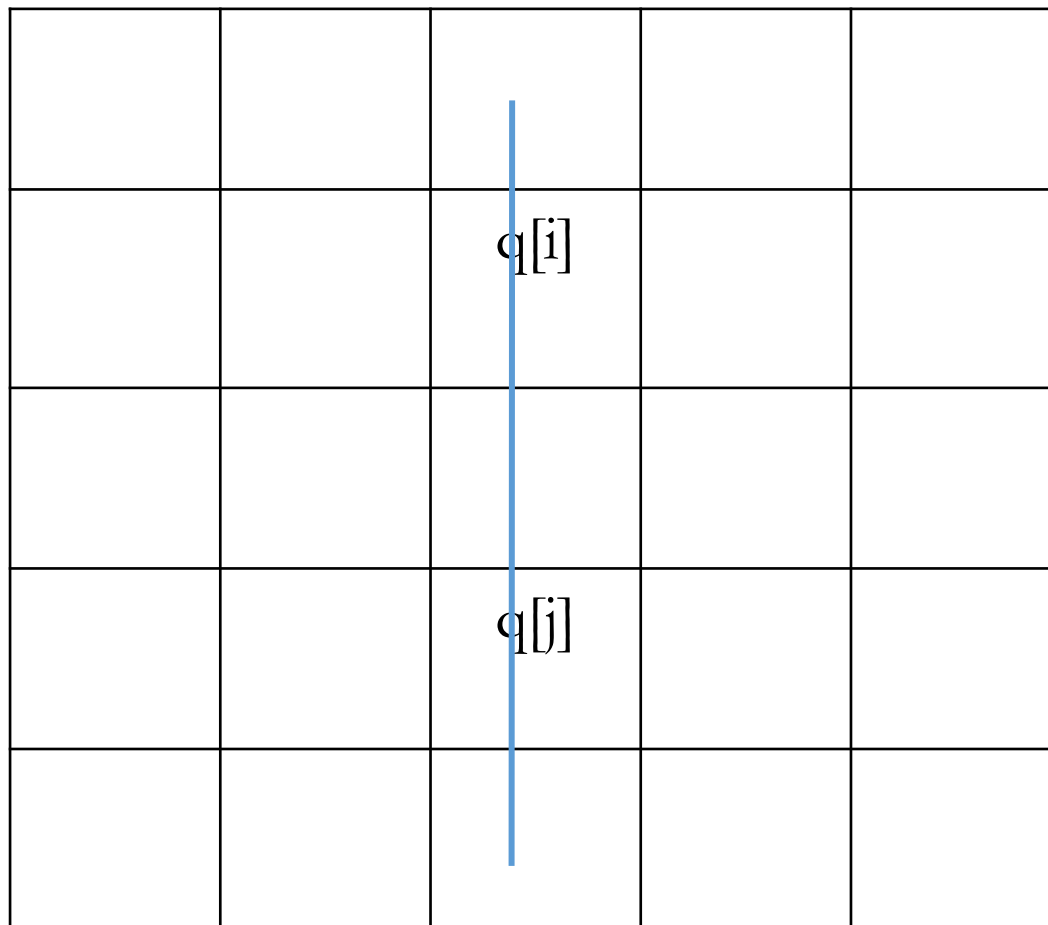
```
void visit(int q[], int i)
{
    i, Nの関係をチェック
    ループ{
        q[i]=?;
        visit(q, ?); }
}
int main(void)
{
    visit(q, 0); // i=0で呼び出す
    return 0;
}
```

Nクイーン問題を解く

- 上記「縦P列の盤面にNクイーンを再帰的に置く」プログラムで $P=N$ にして、解候補 q を求める
- 解候補ひとつひとつ q について、後述のNクイーン配置の条件をチェックする

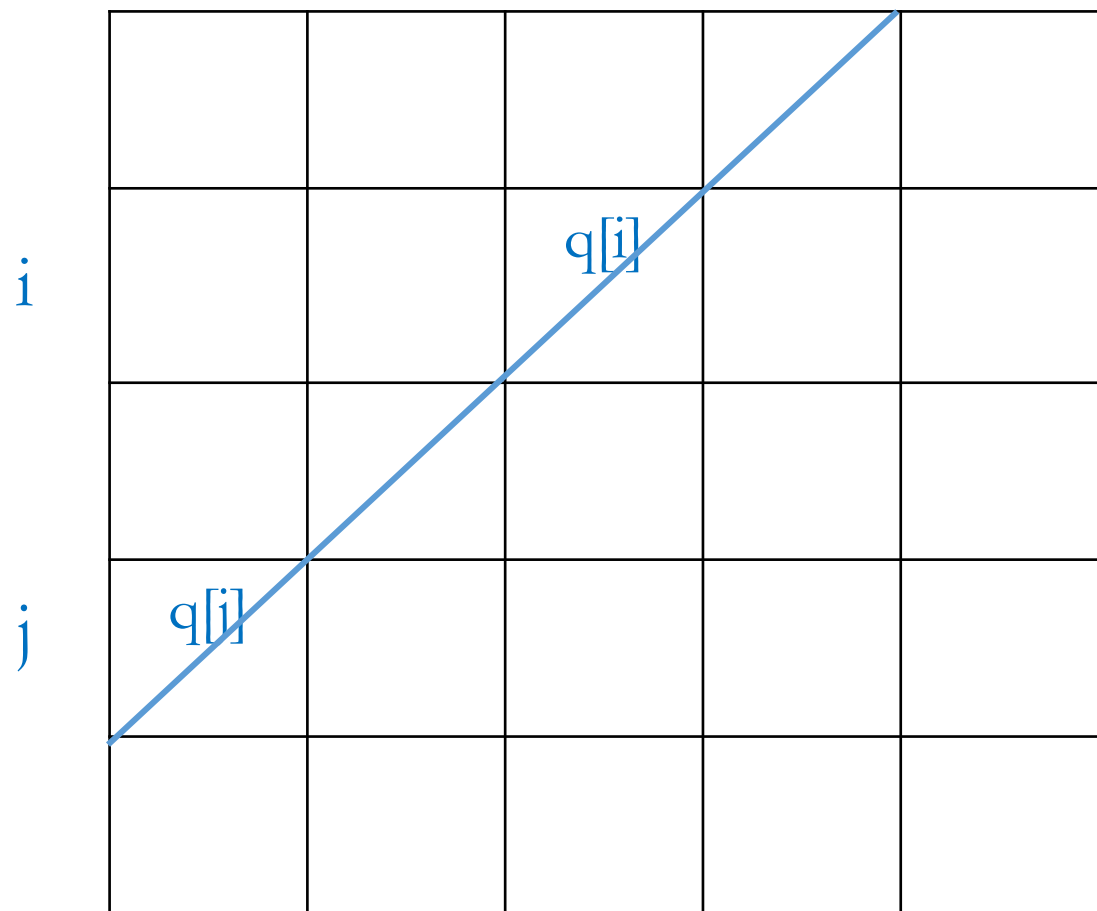
Nクイーン配置条件について

- 縦一直線上にあってはならない $i \neq j$ で $q[i] \neq q[j]$



Nクイーン配置条件について

- 斜め直線上にあってはならない $i \neq j$ で $|q[i]-q[j]| \neq |i-j|$



Nクイーン問題を解くプログラム(基本形)

- void printq(int q[])の定義...配列qの中身を出力
- int check(int q[])の定義...Nクイーンの配置条件でチェックしqが解かどうかをチェック(解なら1を、解でないなら0を返す)

```
void visit(int q[], int i)
{
    if(?) { // iとNの大小関係をチェック
        if(check(q)) printq(q); // 解を出力
        解の個数のカウントなど
    }
    else
        for(int j=0; j<P; j++){
            q[i]=j;
            visit(q, ?);
        }
}
```

```
int main(void)
{
    int q[20];
    visit(q, 0);
    解の個数を出力
}
```

効率をよくしたい

- N=10以上となると、解く時間が長く感じるようになる
- 改良版
 - ある場所に駒置いたら、左斜め直下、直下、右斜め直下への探索を飛ばす
 - そのため、盤面の状態を表す $\text{board}[i][j]$ を設け、置いてよいかの判定に使う。つまり、 (i, j) に駒を置いたらその左斜め直下、直下、右斜め直下の board 値を 0 から 1 に、次を探索し終わった後に、上記値を 0 に戻す
 - 8クイーンの場合、オリジナル版と改良版の解候補の数はそれぞれ 16,777,216, 911,642 となる
- バックトラック法：
 - ある場所に駒を置き、次のステップを見て、解がないとわかればこの駒置きを取り消す

			○				
		×	×	×			

第1回演習課題

- 注意：課題1,2については、課題2ができた時点で提出してください。

第1回演習課題

1. 講義資料の「盤面が縦1列のプログラム」の関連内容を参考に、1次元配列 $q[0], \dots, q[7]$ に値0を再帰的にセットするプログラム([ex01-1row.c](#))を作成しなさい。ただし、値0をセットする度に q の中身を出力する。以下実行例を示す。なおここでは、 q の各要素が値-1で初期化されている。

実行例:

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


第1回演習課題

2. 講義資料の「盤面が縦P列のプログラム」の関連内容を参考に、 $N \times P$ の盤面上にN個のクイーンを配置する解の候補を求めるプログラム(ex01-NxP.c)を作成しなさい。ただし、個々のクイーンは2次元位置情報($i, q[i]$)で出力すること。なお、($i, q[i]$)は*i*行目、 $q[i]$ 列目にクイーンを置く意味である。

実行例

$N \times P$ のN, Pを入力(20以下): 3 2 ← クイーン3個、3x2の盤面

(0,0) (1,0) (2,0)

(0,0) (1,0) (2,1)

(0,0) (1,1) (2,0)

(0,0) (1,1) (2,1)

(0,1) (1,0) (2,0)

(0,1) (1,0) (2,1)

(0,1) (1,1) (2,0)

(0,1) (1,1) (2,1)

CNT: 8

第1回演習課題

3. 講義資料の「Nクイーン問題を解くプログラム(基本形)」及びその関連内容を参考に、8クイーン問題の解を求めるプログラム(ex01-8queen.c)を作成しなさい。ただし、すべての解 q と解候補の数と解の数を8queen.lisというファイルに保存すること

実行例

```
./a.out > 8queen.lis
```

```
tail 8queen.lis (ファイルの末尾のデータを画面上に表示させる)
```

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

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

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

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

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

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

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

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

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

```
CNT_ALL: 16777216, CNT: 92
```

第1回演習課題

4. (**発展問題**) 講義資料の「Nクイーン問題を解くプログラム(基本形)」及びその関連内容を参考に、8クイーン問題の解を求める、**バックトラック法ではない改良版プログラム(ex01-8queen-adv.c)**を作成しなさい。ただし、すべての解 q と解候補の数と解の数を8queen.lisというファイルに保存すること

実行例

```
./a.out > 8queen.lis
```

```
tail 8queen.lis (ファイルの末尾のデータを画面上に表示させる)
```

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

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

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

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

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

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

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

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

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

```
CNT_ALL: 911642, CNT: 92
```