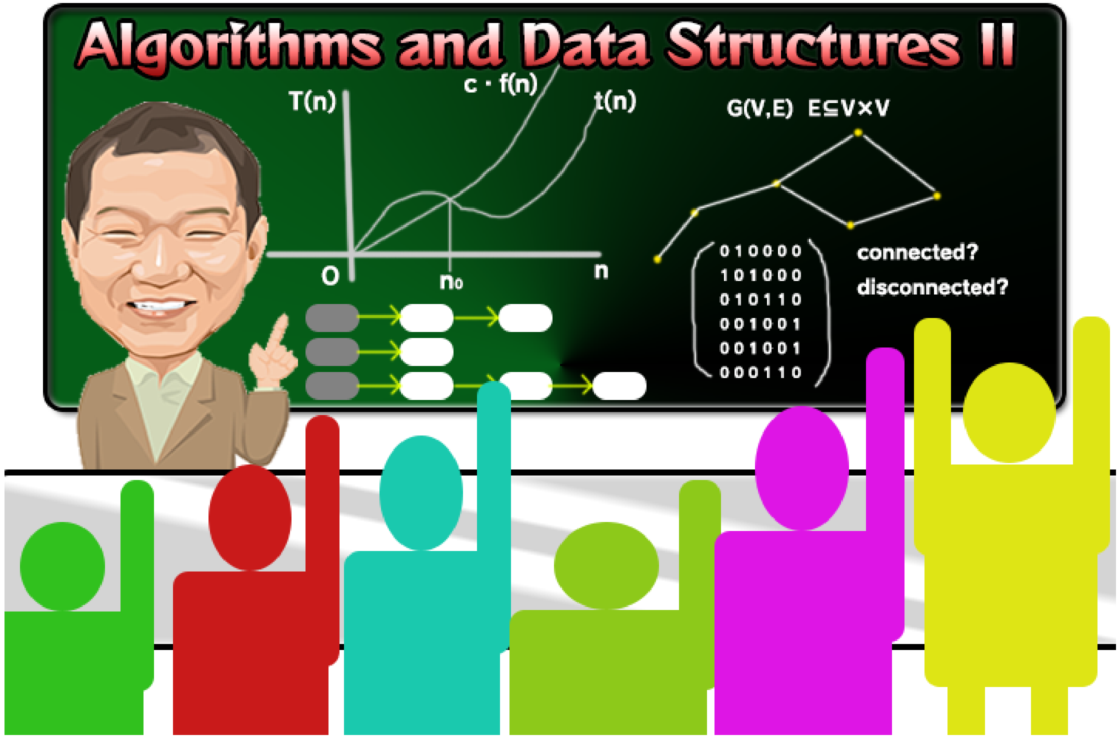


アルゴリズムとデータ構造 II
Algorithms and Data Structures II

Exercise 11. Algorithm Design Techniques IV

Exercise 08-11のために

アルゴリズム設計のためのストラテジー
（問題を解くための、広い意味での解法の考え方）

- Greedy Algorithm (Ex08)
- Divide and Conquer (Ex09)
- Dynamic Programming (Ex10)
- **Backtracking (Ex11)**

[Ex11] Backtracking (バックトラック)

- ある一定の操作を繰り返して解の探索を行う
- 解の探索が不能になったら、他の探索が可能かどうか
  後戻りすることで調べる
- 他の探索経路が見つかったら、そちらへ向かって解の探索を再開する
- （類似）Depth First Search

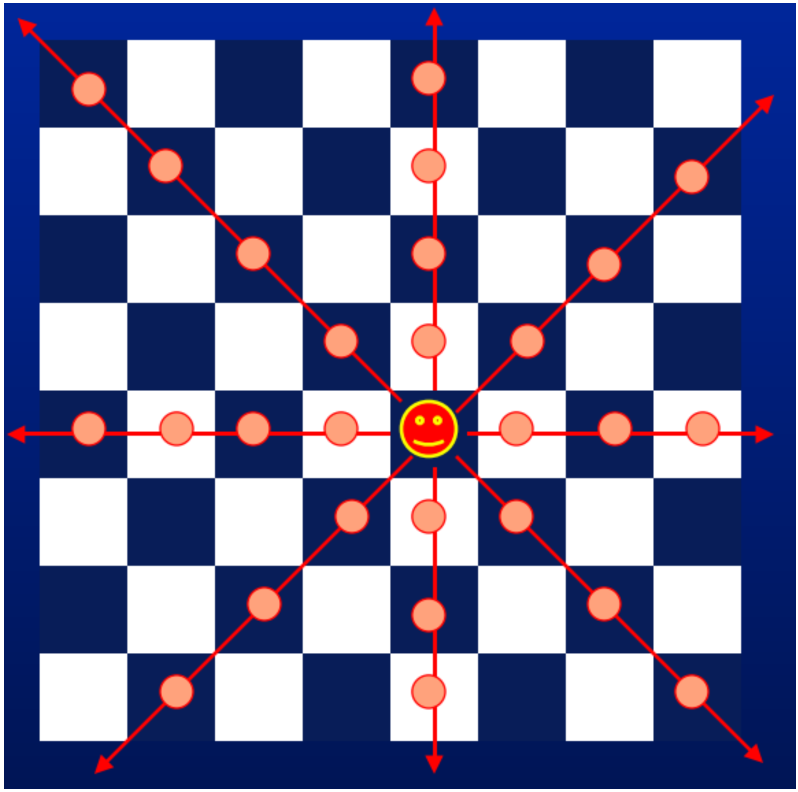
- 講義では N-queen problem を題材にしている

N-queen Problem

- N×Nのチェス盤上にチェスの Queenの駒をN個置く
- ただし、各々のQueenの権力が及ぶ場所が重なってはいけない
- Queen の力は自自身から 縦・横・斜めに向かって伸びる

Nが多くなると、考えられる解の
パターン数は非常に大きくなる。
⇒ 人間が手で解を求めるのは困難

バックトラックによって解を求める。



N-queen Problem

- 8-queen の場合で考える

【基本的な戦略】

・１行目（盤面の上側の横列）から順にQueenを置く場所を探す

・横方向、斜め右上方向、斜め右下方向のいずれの方向にも既にQueenが置かれていないかを確認する

⇒ 無ければ置ける（次の列へ進む）

⇒ あれば置けない（１つ前の置き方が誤りであったと仮定して、１段階戻る）

８行目まで置き進めたらクリア（解を発見したことになる）

N-queen Problem

- 上の行から順に最下行へ進むように探索

⇒ i 行目の解を探索しているときは、「i-1 行目の解までは確定している」という前提のもとに探索

↑ コードを読むときにはこの意識が重要

横方向、斜め右上方向、斜め右下方向

にQueenがあるか無いかだけに集中すればよいので

それを記録する配列があれば十分となる

N-queen Problem

- 配列 row[i] = j

i 行目の j 列目の場所にQueenがあることを示す
⇒ 1列につき1個しか置かないので、この管理法で十分だ

- 配列 col[j]

j 列目に対して横方向にQueenはいるか？いないか？
「置ける、置けない」だけ分かれば良いので それだけを管理

- 配列 pos[2*N-1]

斜め右上方向にQueenはいるか？いないか？
（column と同じ理屈。 neg も同様）

- 配列 neg[2*N-1]

斜め右下方向にQueenはいるか？いないか？

N-queen Problem

- **置けるか置けないかの判定**

col[j]：j行目にQueenはいるか？
pos[i+j]： i+j番目の斜め右上へ向かう射線上にQueenはいるか？
Neg[i-j+N]： i-j+N-1番目の斜め右下へ向かう射線上にQueenはいるか？

```
int try(int i){
int j;

for (j=0;j<N;j++){
  if (col[j]==FREE && pos[i+j]==FREE && neg[i-j+N-1]==FREE ){
    row[i]=j;
    col[j]=NOT_FREE;
    pos[i+j]=NOT_FREE;
    neg[i-j+N-1]=NOT_FREE;

    if (i>=N-1) return SUCCESS;
    else {
       if (try(i+1)==SUCCESS) return SUCCESS;
       else {
     row[i]=-1;
     col[j]=FREE;
     pos[i+j]=FREE;
     neg[i-j+N-1]=FREE;
       }
    }

  }
 }
return FAIL;
}
```

- **Queenを置く**

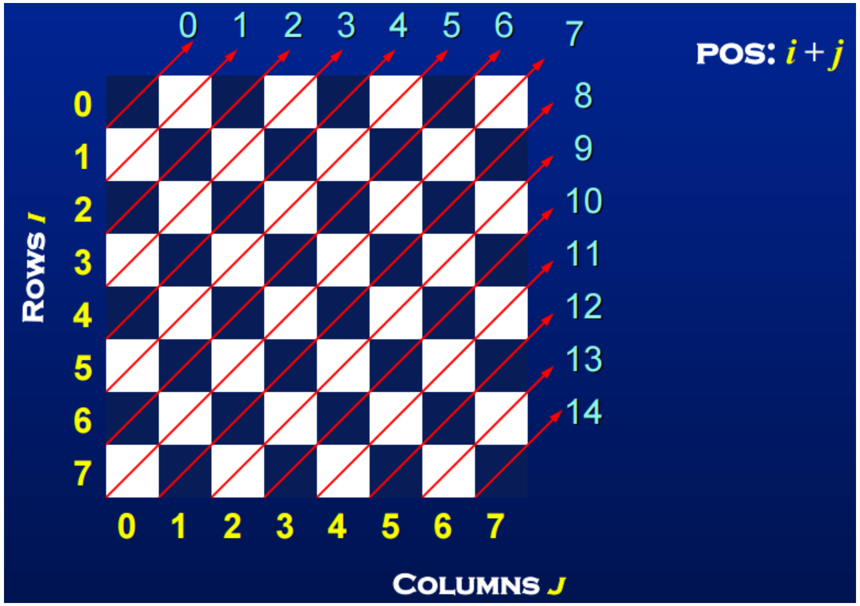
row[i]=j;：i行目はj列目にある
col[j]=NOT_FREE： j列目はもう置くな
pos[i+j]=NOT_FREE：i+j番目の斜め右上へ向かう射線にはもう置くな
neg[i-j+N-1]=NOT_FREE：i-j+N-1番目の斜め右下へ向かう射線にはもう置くな

[i+j番目の斜め、 i-j+N-1番目の斜め とは？]

N-queen Problem (N=8の場合)

- pos[i+j]

斜め右上に向かう射線



「同じ射線上にある盤面上の座標」という関係を考えた時の同値類に相当する

⇒ (5,2)や(4,3)、(0,7)などは

皆 i+j=7番目の右上がりの射線の中に含まれるメンバーである

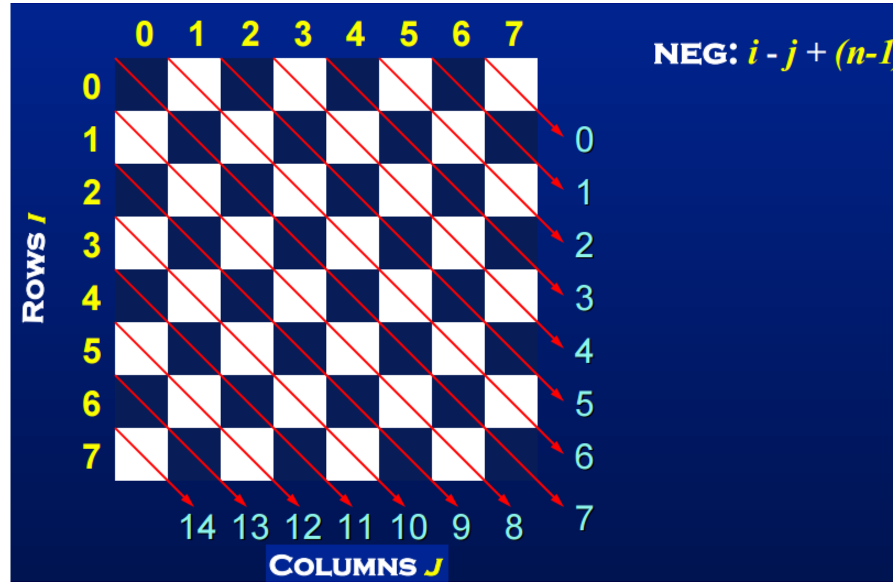
⇒ (4,7)や(5,6)、(6,5)などは

皆 i+j=11番目の右上がりの射線の中に含まれるメンバーである

N-queen Problem (N=8の場合)

- neg[i-j+N-1]

斜め右下に向かう射線



「同じ射線上にある盤面上の座標」という関係を考えた時の同値類に相当する

⇒ (3,3)や(5,5)、(7,7)などは

皆 i-j+N-1=7番目の右下がりの射線の中に含まれるメンバー

⇒ (4,1)や(6,3)、(7,4)などは

皆 i-j+N-1=10番目の右下がりの射線の中に含まれるメンバー

N-queen Problem

Queenを置いたら、column[], pos[], neg[] の対応する要素（位置）を NOT_FREE （もう置けない）にセットする

- 後の列でどこかにQueenを置こうとするときに、

「もう置けない」かどうかは前の置き方によって既に決定済。同じ行、同じ射線上に置かれたかどうかだけを見ればよい。

置けないときには、１手前の段階の探索で「置けなかったことにして」（それぞれのフラグをリセットして）別の探索を行う

N-queen Problem

- これらは、バックトラックとは関係の無い、

N-queen Problemにおける空間計算量(Space Complexity)を減らすためのデータ構造である。

2次元配列で全て管理する実装ももちろん可能だが、「どうせ１行、１列、射線１つあたりに高々１個しか置かないんだから…」という不満に基づく、この問題における最適化のテクニックである。