백트래킹
(Backtracking)

- 해의 형태는 n-tuple ($x_1, x_2, \ldots, x_n$)
  각 x_i는 어떤 유한집합 Si에서 선택됨

- 기준함수(criterion function)인 P(x1, x2,..., xn)를 최대(혹은 최소 혹은 만족하는)화 하는 해를 구하는 문제를 해결하는데 적용하는 방법으로서 효율적인 알고리즘이 존재하지 않을 경우에 사용

- m_i를 S_i의 크기라 하자. 그러면 가능한 후보 해가 $m = m_1 m_2 \ldots m_n$ 개 있다 Backtracking 알고리즘은 문제의 입력에 대한 해의 공간을 체계적으로 탐색함으로서 m보다 매우 작은 횟수의 시도로 해를 찾는다.

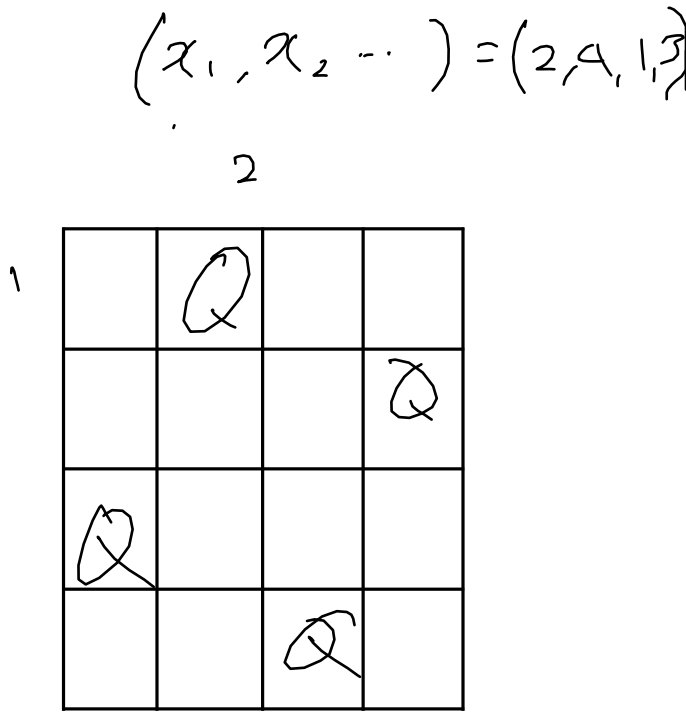
예: N-Queens 문제

- n × n 격자에 다음 조건을 만족하도록 n개의 Queen을 놓아라:
  조건 1) 같은 행에 두개의 Queen이 놓여서는 안된다.
  조건 2) 같은 열에 두개의 Queen이 놓여서는 안된다.
  조건 3) 같은 대각선에 두개의 Queen이 놓여서는 안된다..

- 해: (x1, x2, x3, …xn),
  여기서 xi (1 ≤ i ≤ n) 는 i번째 행에 놓여지는 Queen의 열의 위치로서 1 ≤ xi ≤ n임

  해 공간의 크기 n^n 혹은 n!

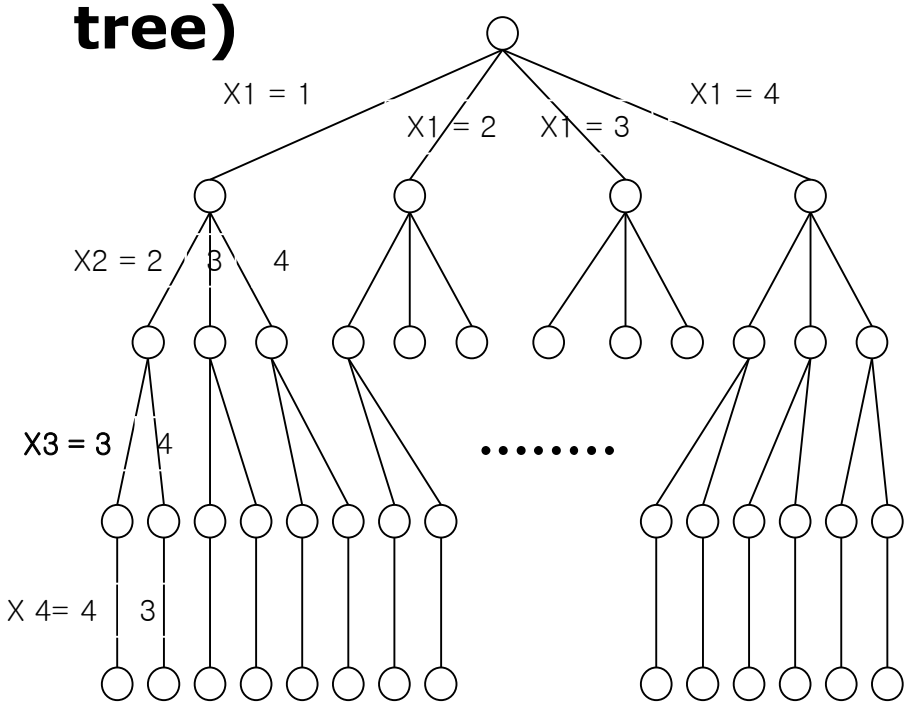
- 예: 4-Queen 문제
  해 (x1, x2, x3, x4) = (2, 4, 1, 3)



백트래킹 알고리즘

- 상태공간트리(State Space Tree)
  모 든 가 능 한 해 (solution space :해공간)을 트리로 구성한 것
- 체 계 적 인 탐 색 방 법 (DFS with bounding function)
  - 문제의 state space(상태공간)을 체계적으로 탐색
  - 탐색시, 해로 도달하지 못하는 것을 발견할 경우, 탐색공간을 prune : Bounding function을 이용하여 탐색이 필요 없는 부분은 제외

- **4-Queens 문 제 의 상태공간트리(state space tree)**



Backtracking (재귀적) 알고리즘

- 구하고자 하는 정답인 해 (x_1, x_2,…, x_n)을 배열 X[1], X[2], … , X[n]에 저장한다.

Algorithm backtrack(x, k, n)
// x[1], x[2], … , x[k-1]가 정해진 상태에서 **x[k], …, x[n]**를 구함

```
for each x[k] such that x[k] ∈T(x[1],…,x[k-1]
  if (Bk(x[1],x[2],…,x[k]))
    if (x[1],…,x[k]) is a path to an answer node,
      output x[1], x[2],…,x[k]
      return
    else if (k < n)
      backtrack(x,k+1,n)
```

- $T(x_1, x_2,…, x_k)$: x_{k+1}에 대한 가능한 값들의 집합t

- B_k: Bounding function (한정 함수)

  만약 root로부터 현재 노드까지의 경로 (x_i, x_2,…, x_k)에 대하여, 해를 계속 찾아볼 필요가 있는지를 판단하는 함수로 참 혹은 거짓

  $B_k(x_i, x_2,…, x_k)$ 이 거짓이면 이 경로는 더 이상 확장할 필요가 없다.

- Backtrack(1)을 호출

N-Queens 문제에 대한 Backtracking 알고리즘

```
Algorithm nQueens(x, k, n)

  // x_k가 가질 수 있는 값들의 집합 T = {1, 2, … , n}
  for i = 1 to n
     if (place(x, k, i)  // Place(x,k,i): Bounding 함수임
          x[k] = i;
     if (k == n)  // 해를 찾은 경우
        x[1],…,x[n]을 출력
     else
        nQueens(x,k+1, n);
```

```
Algorithm place(x, k, i)
   for j = 1 to k−1
      if ((x[j] == i) or abs(x[j] – i) == abs(j – k))
        return false
   return true
```