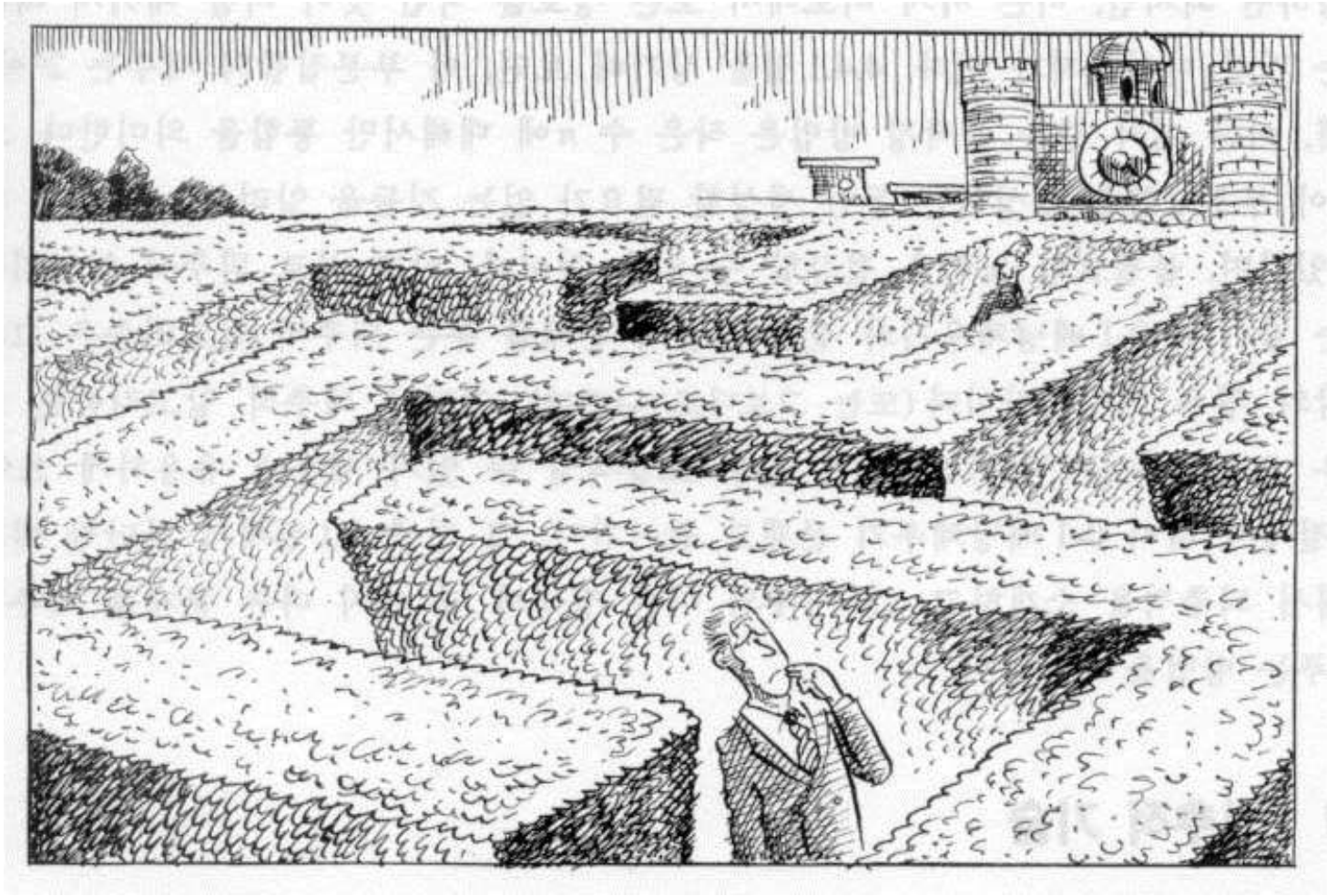
Chap. 5
Backtracking



The General Method

- Many problems which deal with searching for a set of solutions or which ask for an optimal solution satisfying some constraints can be solved using the backtracking formulation.

- The backtrack algorithm has the ability to yield the same answer with for fewer than m trials.

The Basic Idea :

- To build up the same vector one component at a time and to use modified bounding function (or criterion function) $P_i(x_1, \ldots, x_i)$
- to test whether the vector being formed has any chance of success

Advantage: If it is realized that the partial vector $(x_1, x_2, \ldots, x_i)$ can in no way lead to an optimal solution, then $m_{i+1}, \ldots, m_n$ possible test vectors may be ignored entirely.

백트래킹 기법

- 백트래킹 (Backtracking) 기법은 해를 찾는 도중에 '막히면' (즉, 해가 아니면) 되돌아가서 다시 해를 찾아 가는 기법이다
- 백트래킹 기법은 최적화 (optimization) 문제와 결정 (decision) 문제를 해결할 수 있다.
- 결정 문제: 문제의 조건을 만족하는 해가 존재하는지의 여부를 'yes' 또는 'no'가 답하는 문제
  - 미로 찾기
  - 해밀토니안 사이클 (Hamiltonian Cycle) 문제
  - 부분 집합의 합 (Subset Sum) 문제 등

Backtrack 알고리즘의 기본형태

Assume : 1. Solution : X(1..n)

2. $X(k) \in S=\{a_1, a_2, \ldots a_m\}$ for $1 \le k \le n$, where S: finite set

3. Initially, $X(k)= a_0$ for all k, where $a_0 \notin S$

```
Procedure BACKTRACK(n)
  k := 1 ;
  for  i=1 to n do
      X(i) := a0;
  while 1 ≤ k ≤ n do
  {    GETNEXT(k);
      if X(k) = a0                    backtrack
          then k := k−1 ;
          else if k = n ;
                  then OUTPUT(X) ;
                  else  k := k+1;     }
```

```
Procedure GETNEXT(k)

   Let ai be the current value of X(k)

   while  i < m  do
    {   X(k) := a i+1 ;
        i := i + 1;
        if  BOUND(k) = true
            then return ;     }

   if i = m  then X(k) := a0;
```

1) n-Queens 문제

- n-Queen 문제 : n개의 Queen을 서로 상대방을 attack하지 않도록 n × n chess판에 위치시키는 문제

- 서로 상대방을 attack하지 않기 위해서는 같은 행이나, 같은 열이나, 같은 대각선 상에 위치하지 않아야 한다.

8-Queens 문제

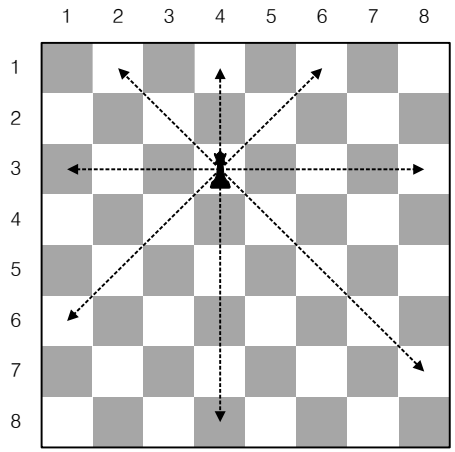
- 후보해의 수: 약 44억개

$$_{64}C_8 = \frac{64!}{8!\,(64-8)!} = 4{,}426{,}165{,}368$$

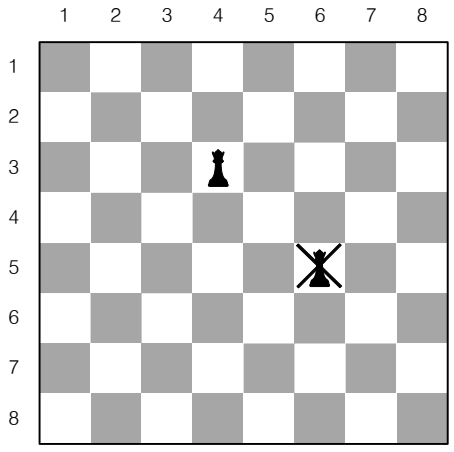
- 실제 해의 수 : 92개

→ 즉, 44억 개가 넘는 후보해들 중 92개를 최대한 효율적으로 찾아내는 것이 관건

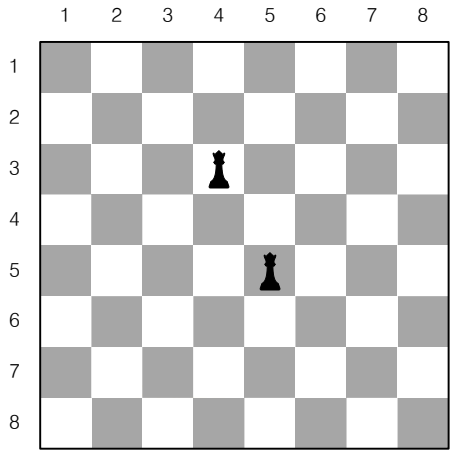
퀸의 공격 범위



서로 공격가능한 배치



서로 공격 불가능한 배치



4-Queens 문제: 1)무작정 알고리즘

- 후보해의 수: ${}_{16}C_4$=1820개이다.
- 그러나, 각 Queen을 각각 다른 행에 할당한 후, (즉, Queen들을 임의로 번호를 붙여 Queen-i는 i행에 할당한다고 가정함) 각 Queen-i를 어느 열에 둘 것인지 차례대로 모두 점검한다. (Assume queen-i is to be placed on row i.)

  ➔ 각 Queen-i은 4개의 열 중에서 위치할 수 있기 때문에, 점검해 보아야 하는 모든 경우의 수는 $4 \times 4 \times 4 \times 4 = 256$가지가 된다.

  즉, ${}_{16}C_4$=1820개 보다는 훨씬 줄어들었음

Explicit constraint

① Queen-i는 i행에 위치하며, i행에 있는 열 중에서 a_i 열에 둔다. 단, a_i 는 집합 S = {1,2,3,4}의 값 중 하나이다.

Implicit constraint

② 어느 두개의 a_i 와 a_k 의 값은 서로 달라야 한다.
즉, 같은 열에 위치하면 안된다.

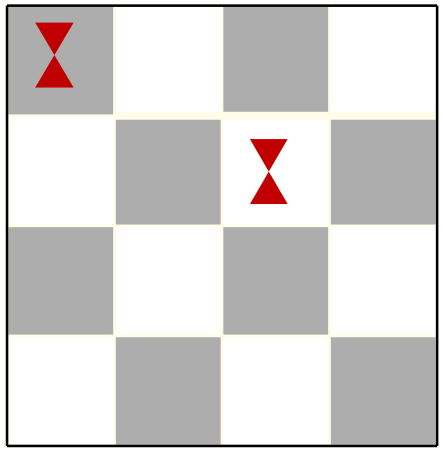
➔ 4! tuples. (즉, n! permutations of n-tuples)

③ 어느 두 Queen도 같은 대각선 상에 있어서는 안된다.

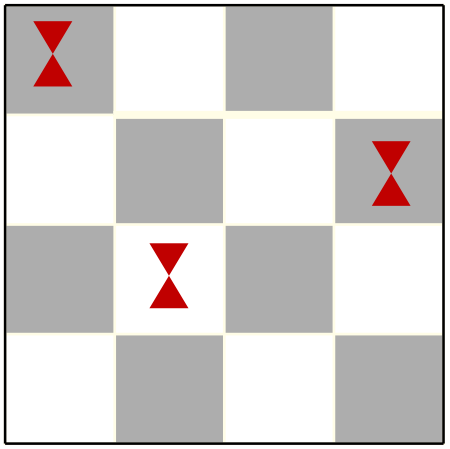
- 다음 그림을 생각해보자.

  같은 행, 열, 대각선이 아닌 장소에 두었으나 해답으로 갈 수 있는 가능성이 없음을 알 수 있다.

??



??



➔ Backtrack~!

잠깐, 복습 . . .

<Recall> Depth-First Search

- root가 되는 노드를 먼저 방문한 뒤, 그 노드의 모든 descendant(후손)들을 차례로 (보통 왼쪽에서 오른쪽으로) 방문한다.

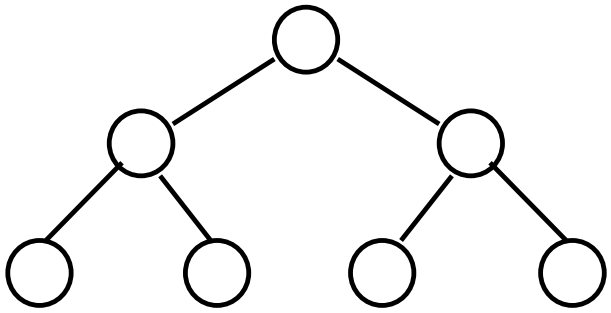
  ➔ 트리에서는 preorder tree traversal과 같다.

```
void DFS(node v) {
    node u;
    visit[v] := 1;
    for (each node u which is adjacent to v)
        DFS(u)
}
```

상태공간트리(State Space Tree)

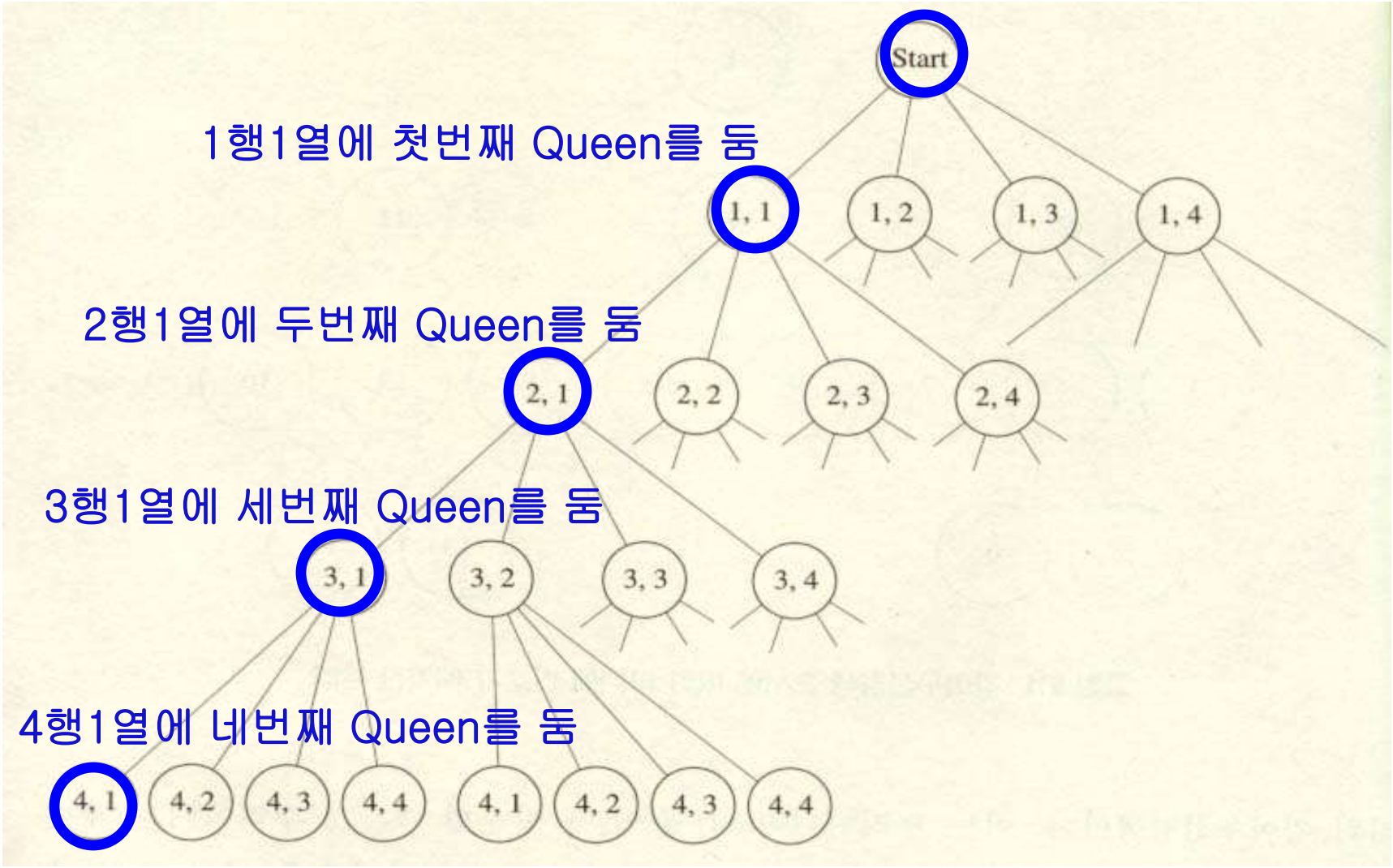
- 문제해결 과정의 중간 상태를 각 노드로 나타낸 트리
  - ✓ 각 노드들을 problem state (혹은 state)라 함.
  - ✓ Candidate Solution(해답후보) : 루트에서 리프까지의 모든 경로
  - ✓ Answer States : 루트에서 solution states S로의 경로가 solutions의 값이 되는 tuple

- 상태공간 탐색 기법
  - ✓ 백트래킹(5장)
  - ✓ 분기한정(6장)



<State Space Tree>

DFS(깊이우선탐색)으로 형성되는 State Space Tree(상태공간트리)



DFS로 형성되는 state space tree :

- Root 노드에서 leaf 노드까지의 경로는 해답후보 (candidate solution)가 되는데, 깊이우선검색을 하여 그 해답후보 중에서 해답을 찾을 수 있다.

  ➔ 그러나 이 방법을 사용하면 해답이 될 가능성이 전혀 없는 노드의 descendant (후손노드)들도 모두 검색해야 하므로 비효율적이다.

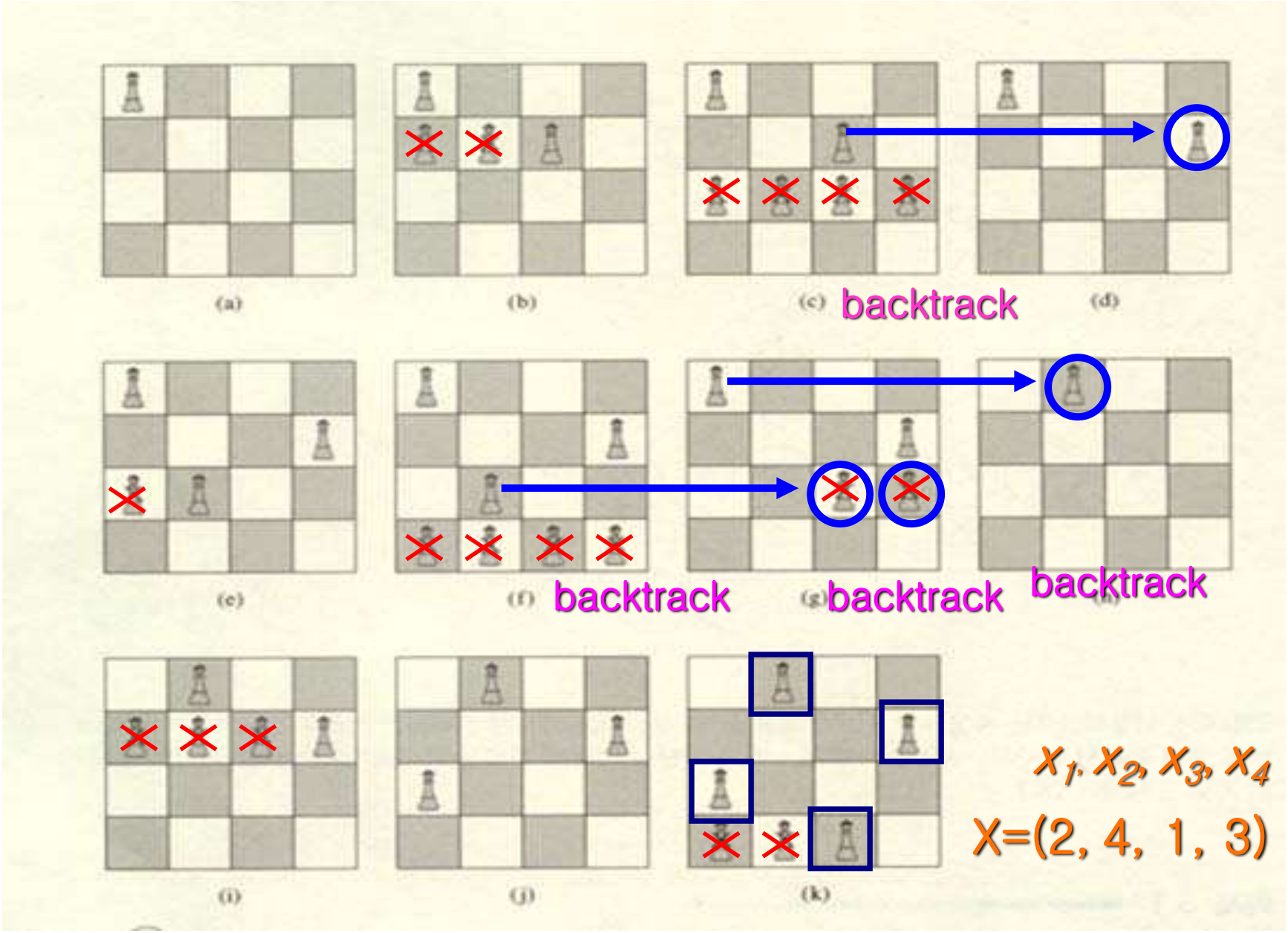
4-Queens 문제: 2) Backtrack 기술

- 노드의 유망성:
  - ✓ 전혀 해답이 나올 가능성이 없는 노드는 non-promising (유망하지 않다)이라 하고,
  - ✓ 그렇지 않으면 promising(유망하다)이라 한다.
- Back Track이란?
  - ✓ 어떤 노드의 유망성을 점검한 후, 유망하지 않다고 판정이 되면 그 노드의 부모노드(parent)로 돌아가서 (➔backtracking) 다음 후손노드에 대한 검색을 계속하게 되는 절차.

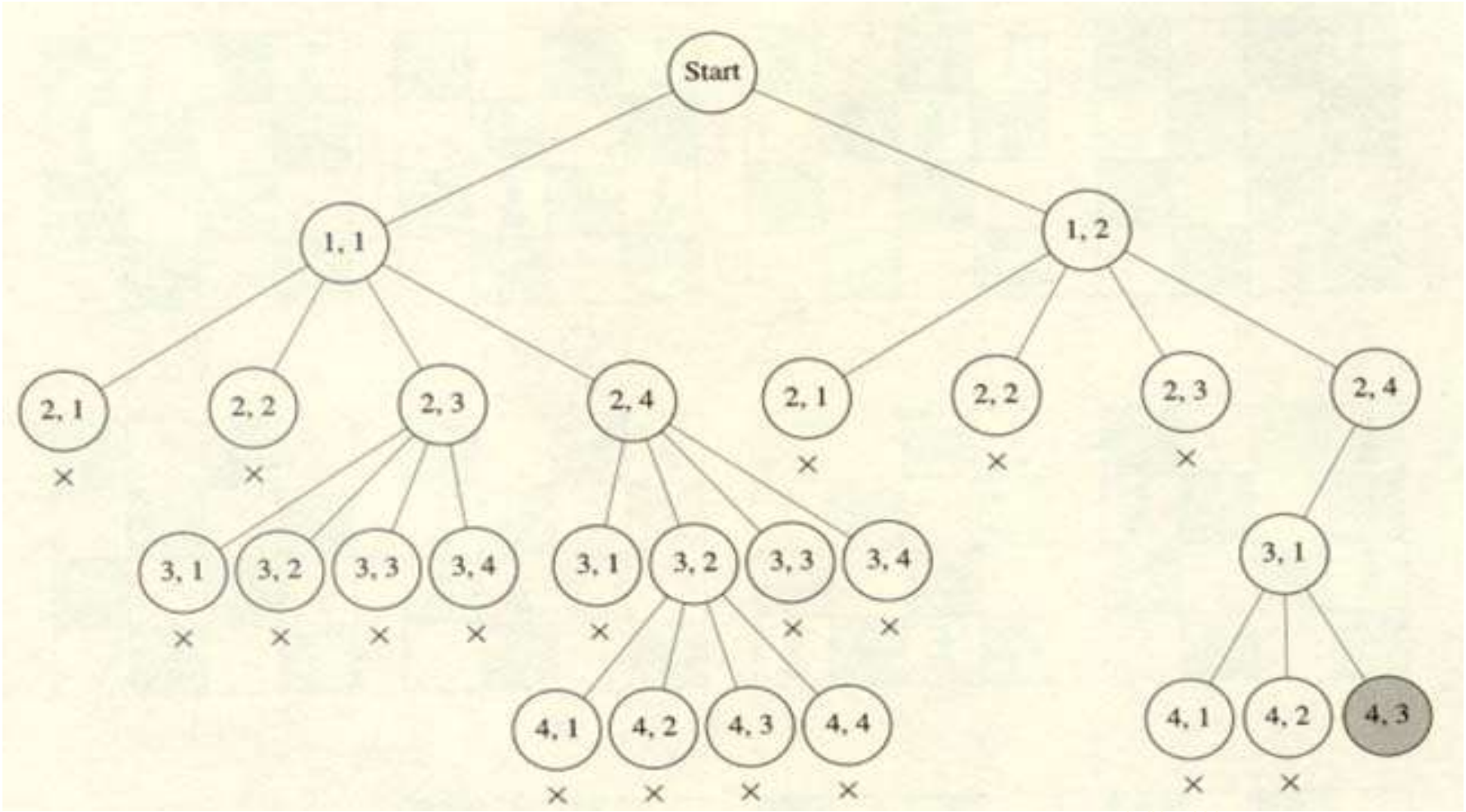
Backtracking 알고리즘의 개념

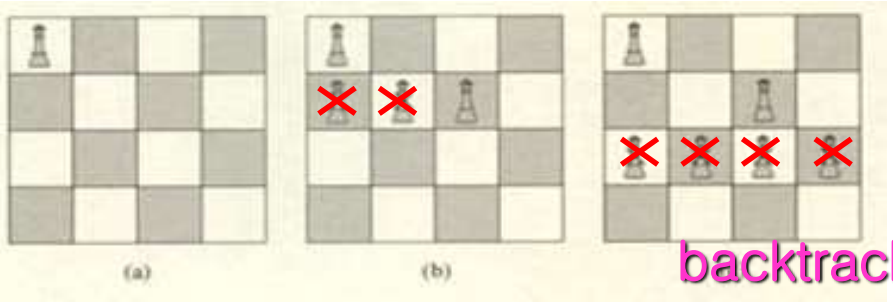
- State space tree에서 깊이우선검색을 실시하는데,
  - ✓ Non-promising 노드들은 가지를 쳐서(pruning) 검색을 하지 않으며,
  - ✓ Promising노드에 대해서만 그 노드의 children을 검색한다.

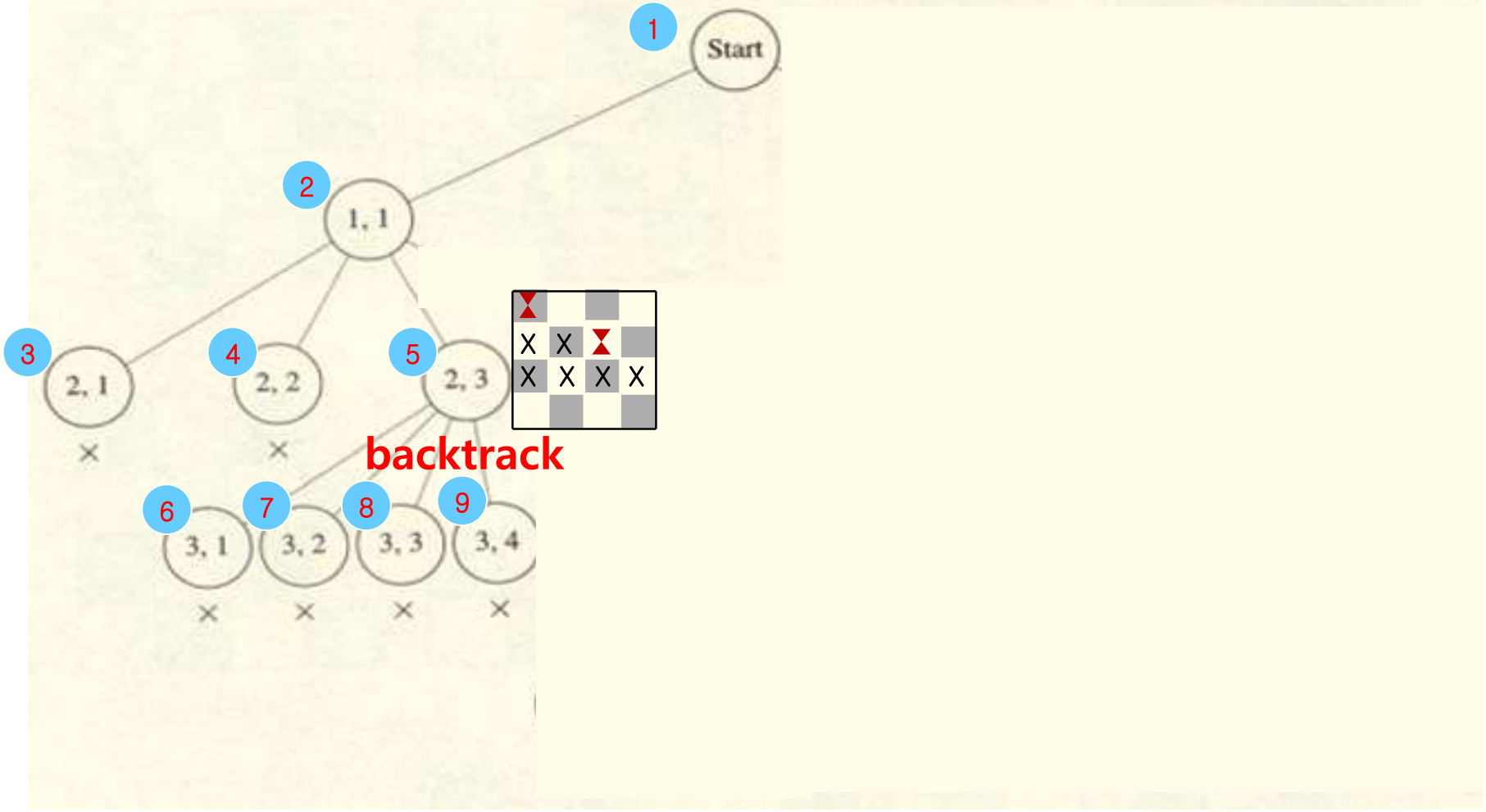
- 다음과 같은 절차로 진행
  1) State Space Tree의 깊이우선검색을 실시
  2) 각 노드가 promising한지를 점검
  3) 만일 그 노드가 non-promising이면, 그 노드의 부모노드로 돌아가서 검색을 계속

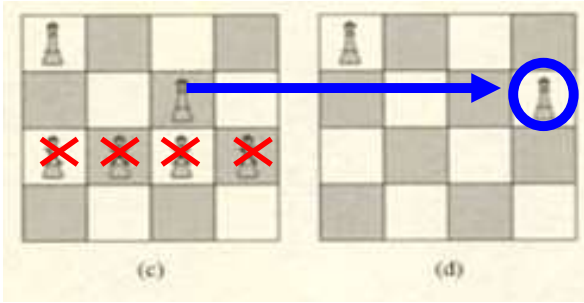

(a)
(b)
(c)
backtrack
(d)
(e)
(f)
backtrack
(g)
backtrack
backtrack
(i)
(j)
(k)
x_1, x_2, x_3, x_4
X=(2, 4, 1, 3)

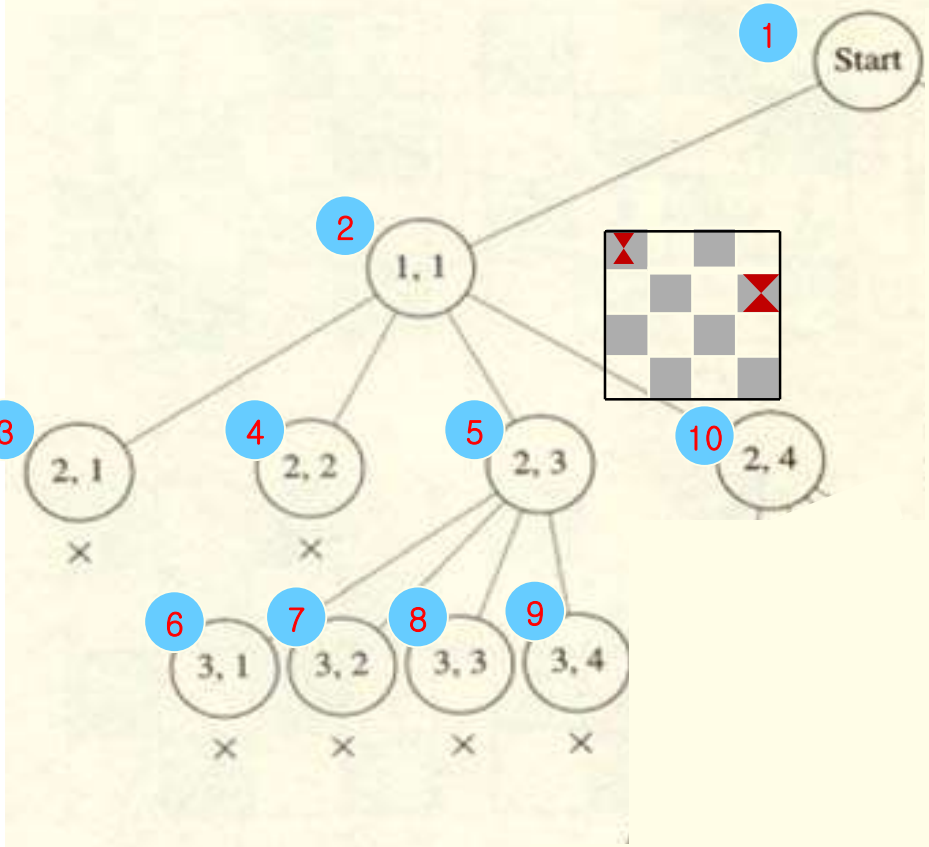
Backtracking 알고리즘으로 형성되는 state space tree

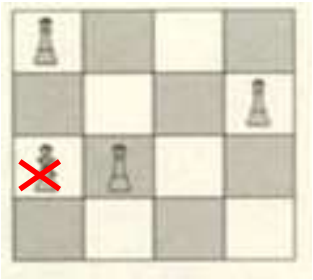


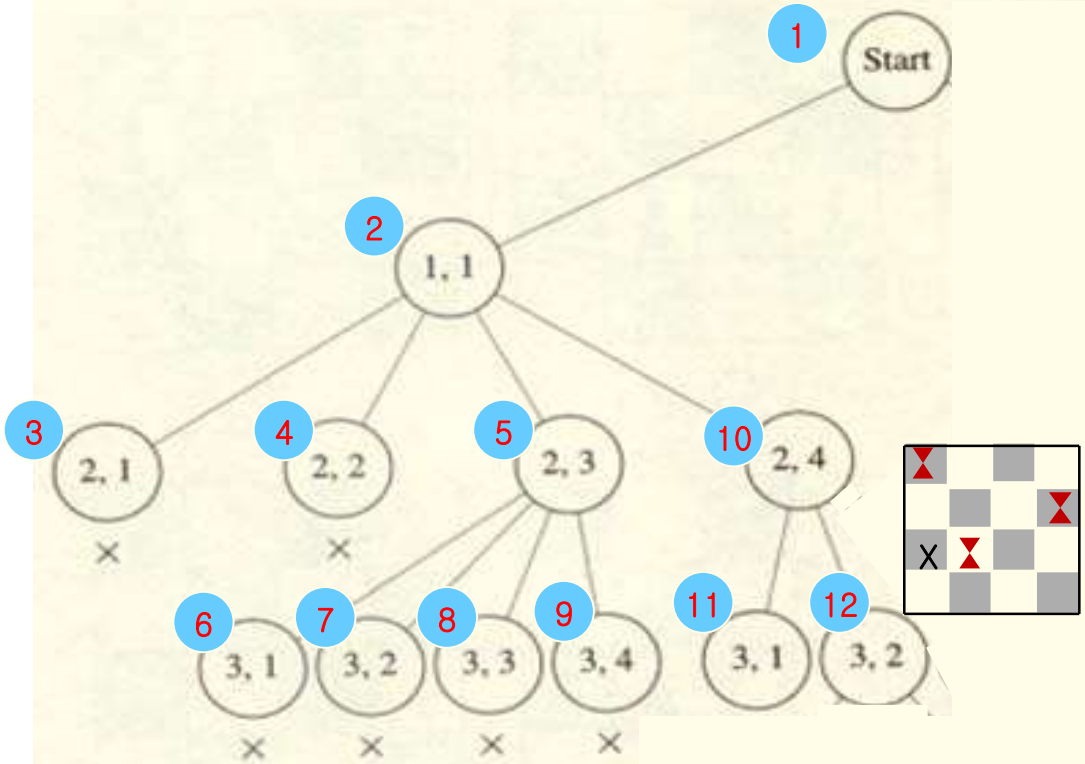

(a)
(b)
backtrack

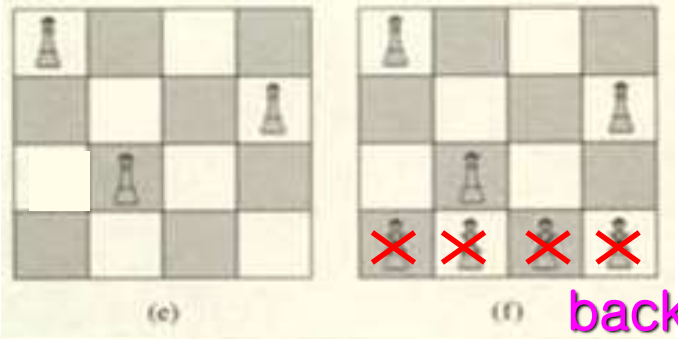

1
Start
2
1, 1
3
2, 1
4
2, 2
5
2, 3
backtrack
6
3, 1
7
3, 2
8
3, 3
9
3, 4


(c)
(d)

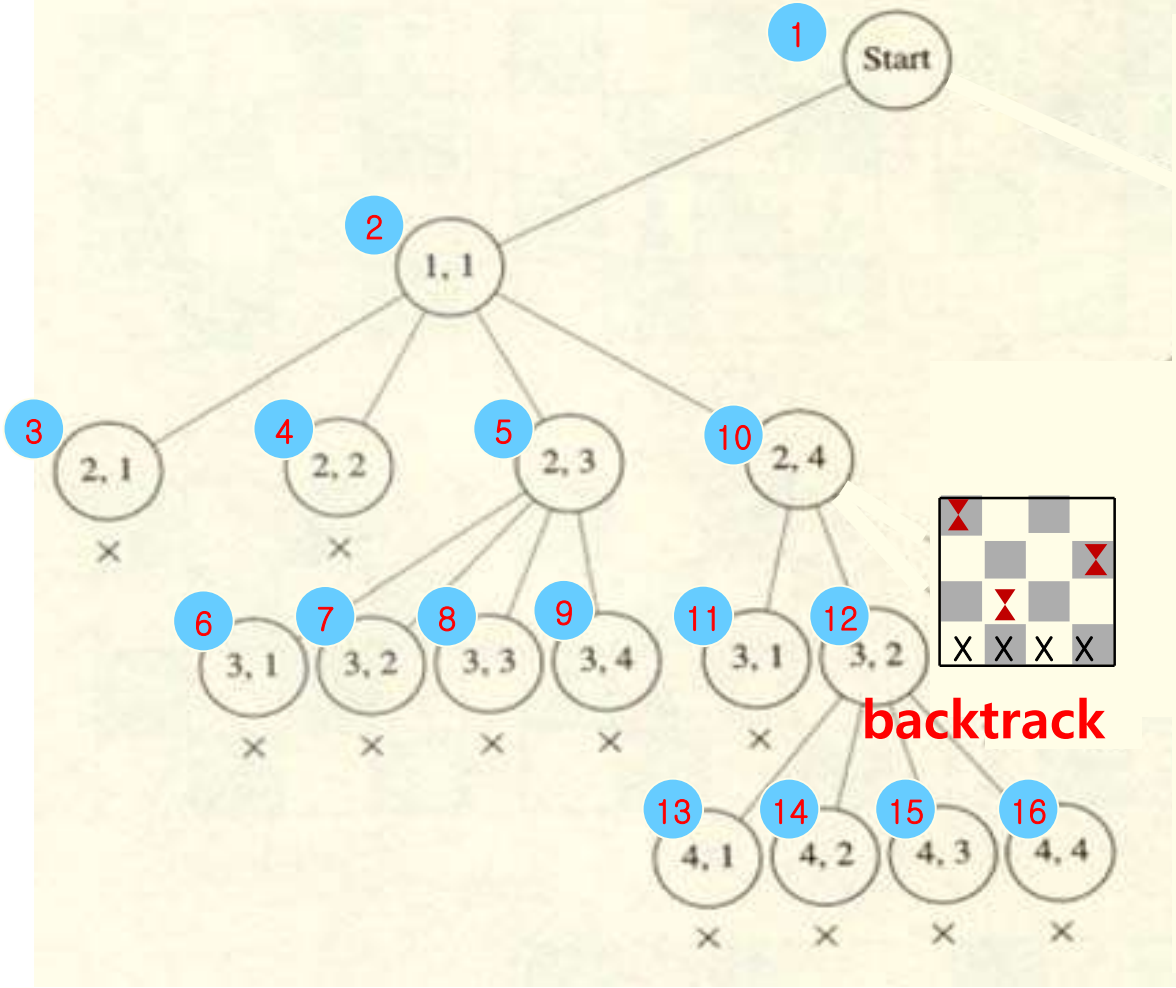

1
Start
2
1, 1
3
2, 1
4
2, 2
5
2, 3
10
2, 4
6
3, 1
7
3, 2
8
3, 3
9
3, 4



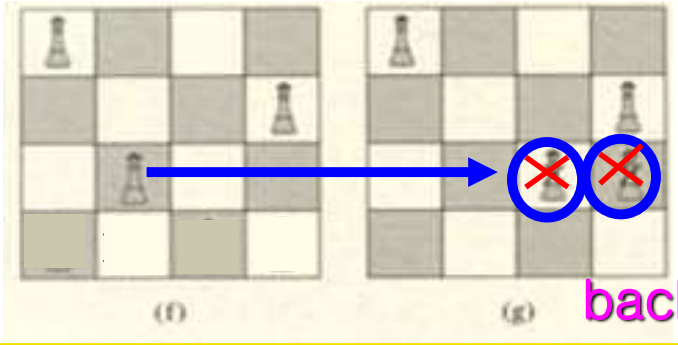

1
Start
2
1, 1
3
2, 1
4
2, 2
5
2, 3
10
2, 4
6
3, 1
7
3, 2
8
3, 3
9
3, 4
11
3, 1
12
3, 2
X



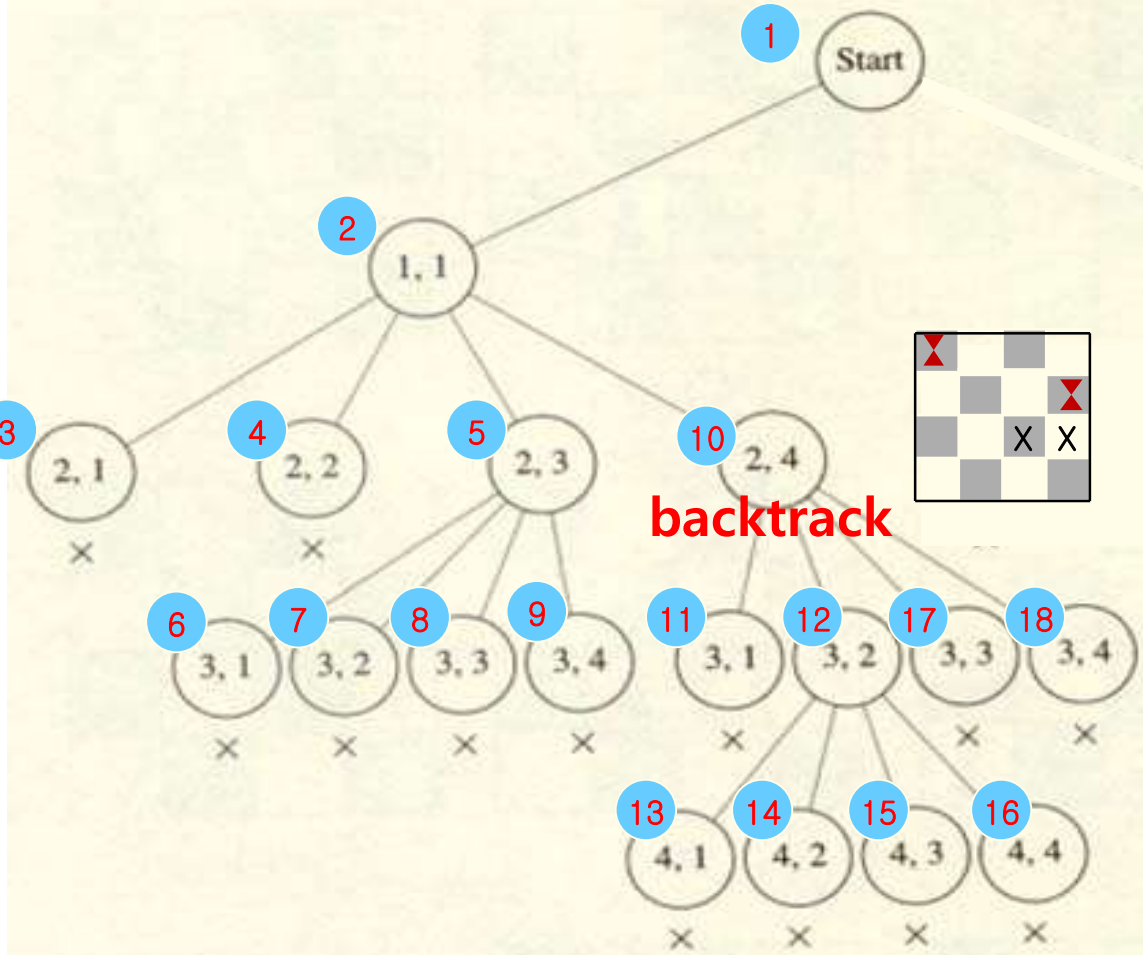
backtrack

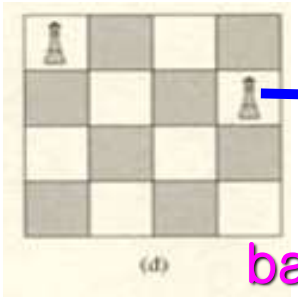


backtrack

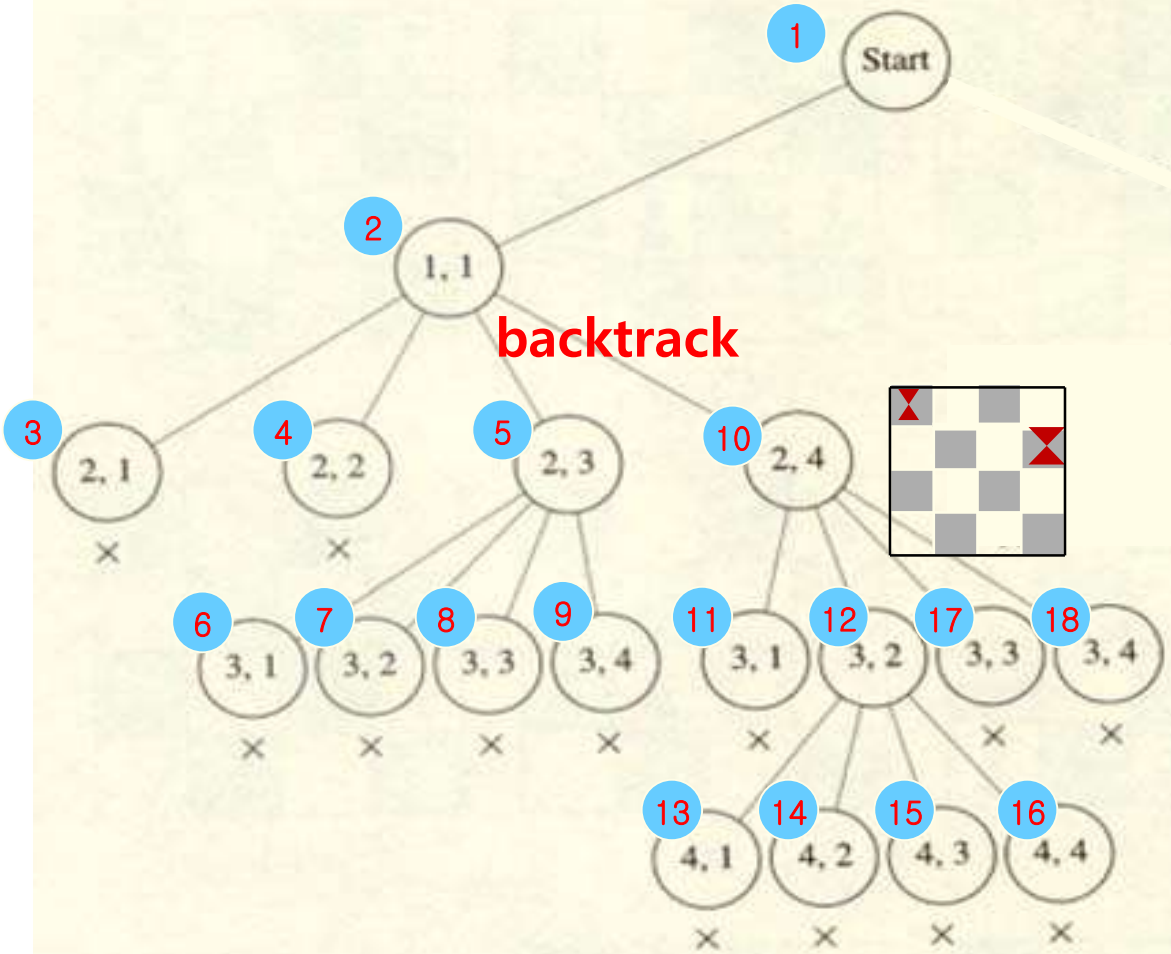

(f)
(g)

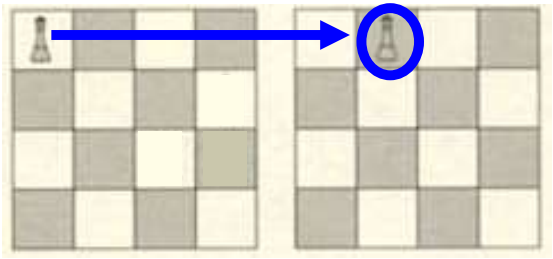
backtrack

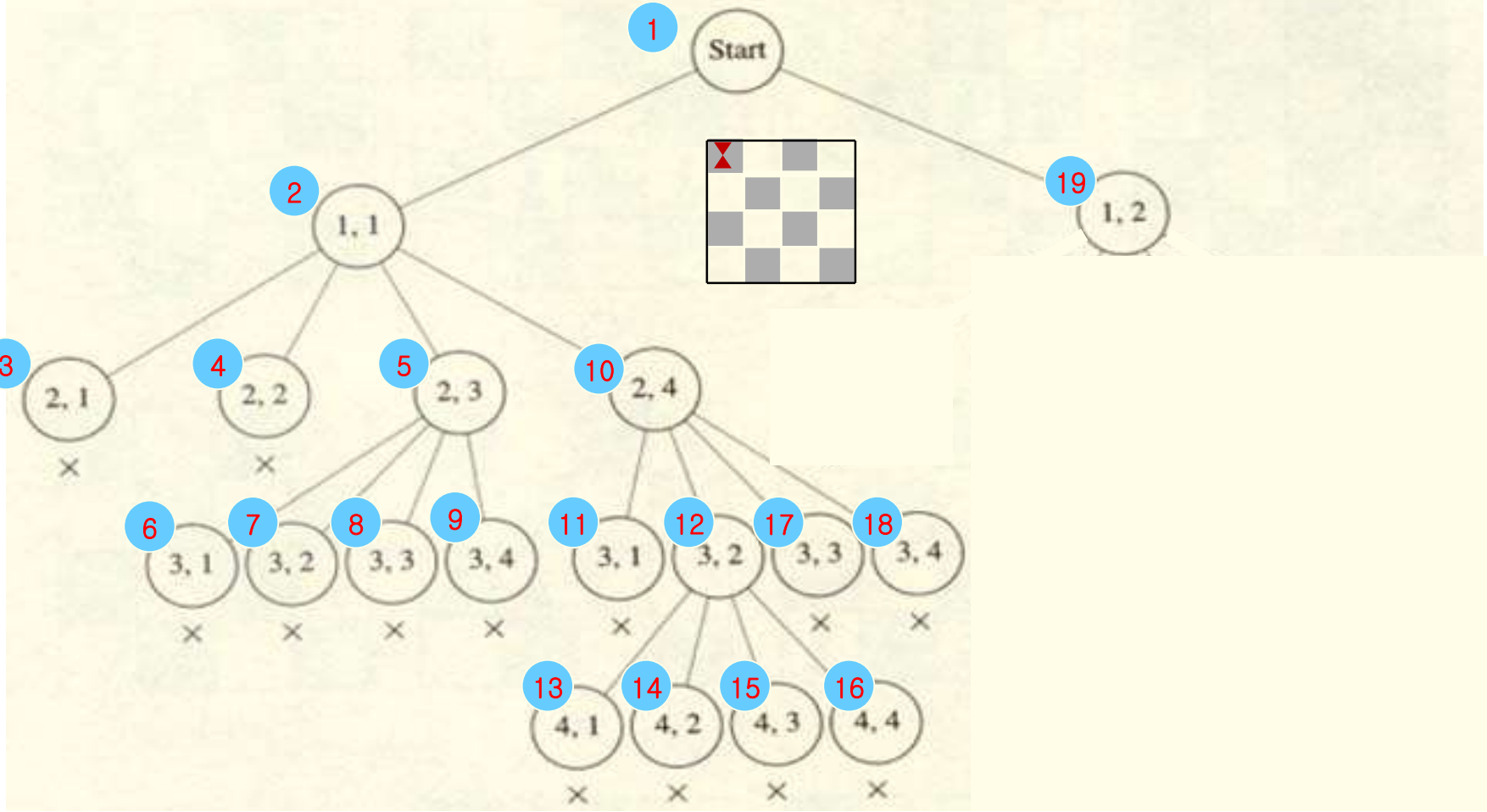

1
Start
2
1, 1
3
2, 1
4
2, 2
5
2, 3
10
2, 4
backtrack
6
3, 1
7
3, 2
8
3, 3
9
3, 4
11
3, 1
12
3, 2
17
3, 3
18
3, 4
13
4, 1
14
4, 2
15
4, 3
16
4, 4

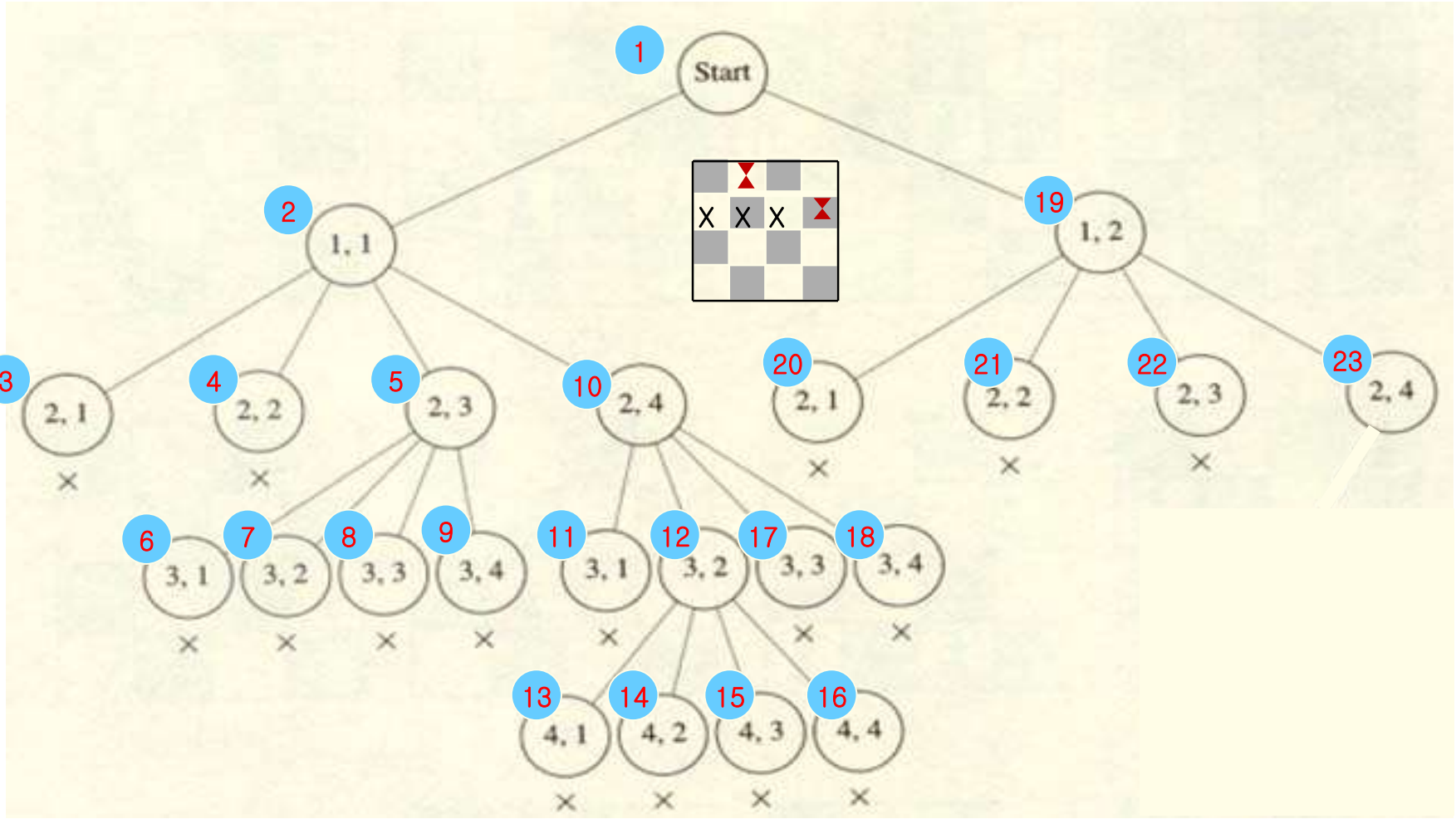

(d)

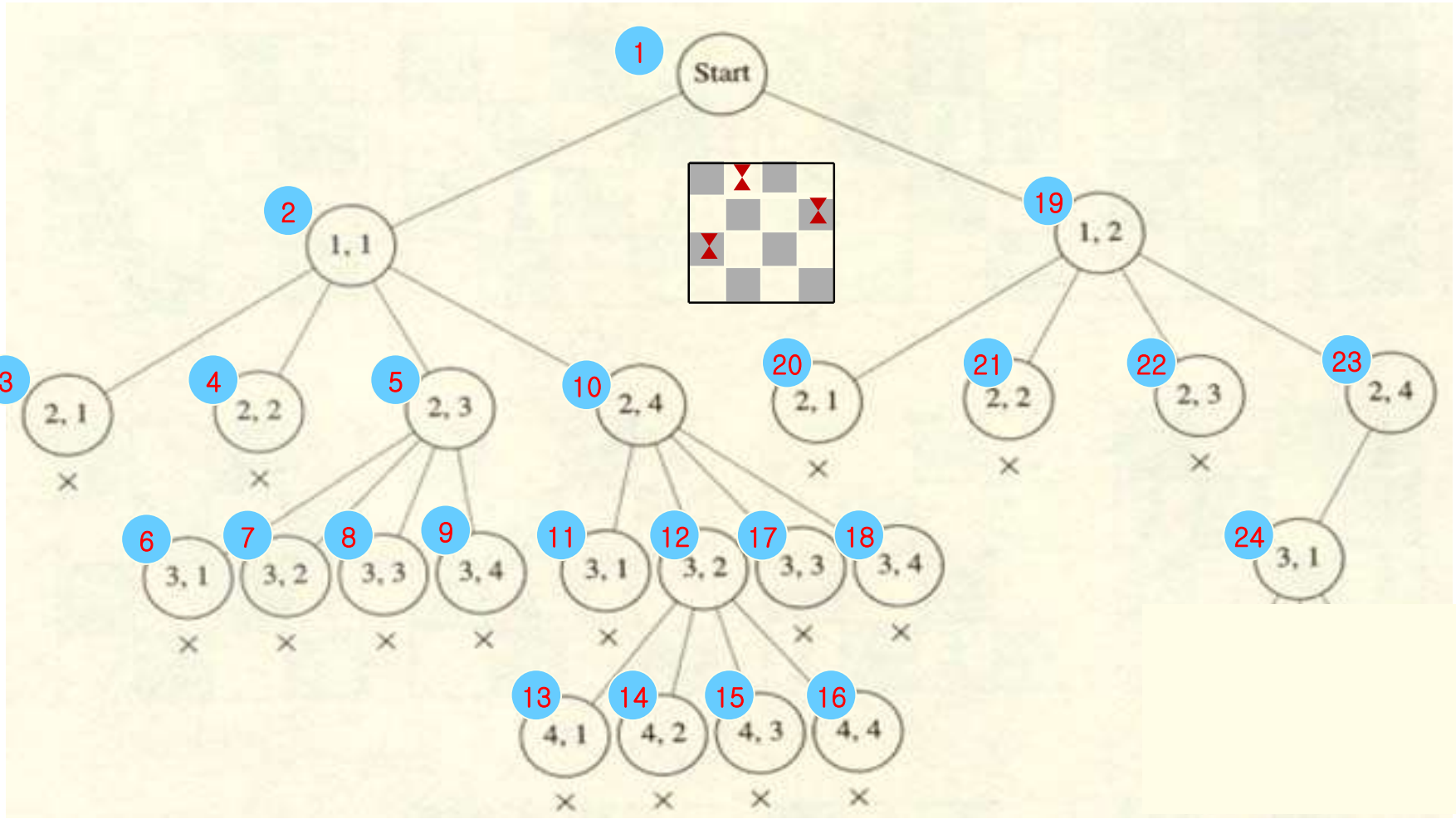
backtrack

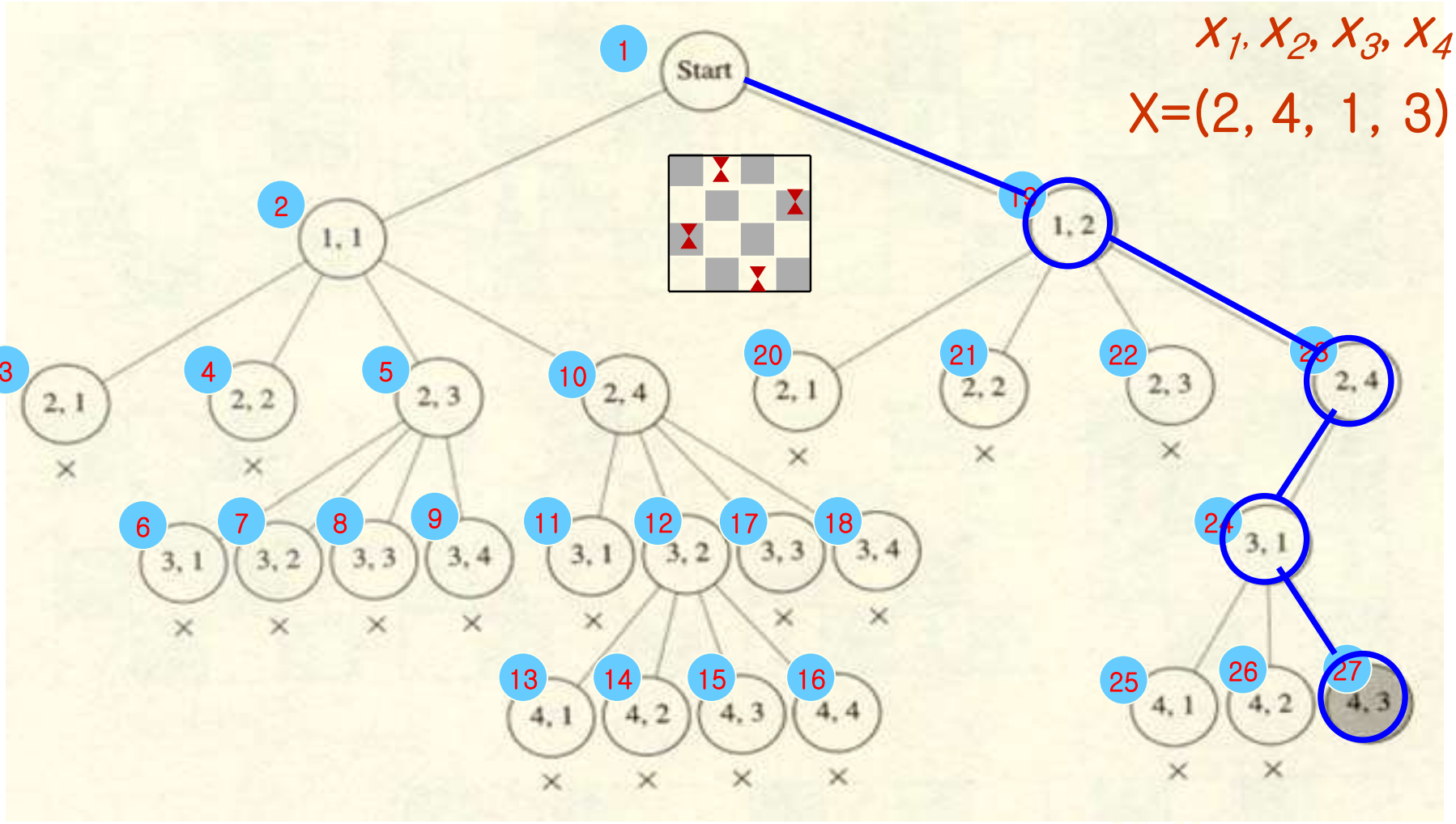

1
Start
2
1, 1
backtrack
3
2, 1
4
2, 2
5
2, 3
10
2, 4
6
3, 1
7
3, 2
8
3, 3
9
3, 4
11
3, 1
12
3, 2
17
3, 3
18
3, 4
13
4, 1
14
4, 2
15
4, 3
16
4, 4




1
Start
2
1, 1
19
1, 2
3
2, 1
4
2, 2
5
2, 3
10
2, 4
6
3, 1
7
3, 2
8
3, 3
9
3, 4
11
3, 1
12
3, 2
17
3, 3
18
3, 4
13
4, 1
14
4, 2
15
4, 3
16
4, 4


1
Start
2
1, 1
19
1, 2
3
2, 1
4
2, 2
5
2, 3
10
2, 4
20
2, 1
21
2, 2
22
2, 3
23
2, 4
6
3, 1
7
3, 2
8
3, 3
9
3, 4
11
3, 1
12
3, 2
17
3, 3
18
3, 4
13
4, 1
14
4, 2
15
4, 3
16
4, 4


1
Start
2
1, 1
19
1, 2
3
2, 1
4
2, 2
5
2, 3
10
2, 4
20
2, 1
21
2, 2
22
2, 3
23
2, 4
6
3, 1
7
3, 2
8
3, 3
9
3, 4
11
3, 1
12
3, 2
17
3, 3
18
3, 4
24
3, 1
13
4, 1
14
4, 2
15
4, 3
16
4, 4


x_1, x_2, x_3, x_4
X=(2, 4, 1, 3)
1
Start
2
1, 1
3
2, 1
4
2, 2
5
2, 3
6
3, 1
7
3, 2
8
3, 3
9
3, 4
10
2, 4
11
3, 1
12
3, 2
13
4, 1
14
4, 2
15
4, 3
16
4, 4
17
3, 3
18
3, 4
1, 2
20
2, 1
21
2, 2
22
2, 3
2, 4
3, 1
25
4, 1
26
4, 2
27
4, 3

Bounding function

- Chessboard의 각 cell (각 •)에 주소(address)를 붙인다.

행 열

- 주소: A(1:n, 1:n)
- 같은 diagonal(\)에 있는 element들은 (row - column)의 값이 같다.

  예) A(1,2), A(2,3), A(3,4)는 같은 대각선 \ 상에 있다.

- 마찬가지로, 같은 diagonal (/)에 있는 element들은 같은 (row + column) 값을 갖는다.

  예) A(1,3), A(2,2), A(3,1)는 같은 대각선 / 상에 있다.

Bounding function (앞 페이지에서 계속)

만약 어떤 두 queen의 위치가 각각 A(a, b), A(c, d)라 두자.

If a-b=c-d or a+b=c+d, they are on the same diagonal.

a-b=c-d or a+b=c+d

⇔ b-d=a-c or b-d=c-a=-(a-c)

⇔ b-d = ±(a-c) =| a-c | ∴ | b - d | = | a – c |

➔ Queen이 같은 대각선 상에 있다면,

| b - d | = | a – c | 이 된다.

★ k-th queen은 i-th queen ($1 \le i \le k-1$)의 값 X(i)와 다음 식이 성립하면 현재 위치하려는 자리 X(k)에 둘 수 없다.

$$| i - k | = | X(i) - X(k) |$$

DFS vs. Backtracking

- 4-Queen 문제에서 검색하는 노드 개수의 비교
  - ✓ 순수한 깊이우선검색 = 155 노드
  - ✓ backtracking = 27 노드

n-Queens 문제의 분석

- Promising 노드의 개수를 정확하게 구하기 위한 유일한 방법은 실제로 알고리즘을 수행하여 구축된 state space tree의 노드의 개수를 세어보는 수 밖에 없다.
- 그러나 이 방법은 진정한 분석 방법이 될 수 없다. 왜냐하면 분석은 알고리즘을 실제로 수행하지 않고 이루어져야 하기 때문이다.

Monte Carlo 기법을 사용한 Backtracking 알고리즘의 수행시간 추정

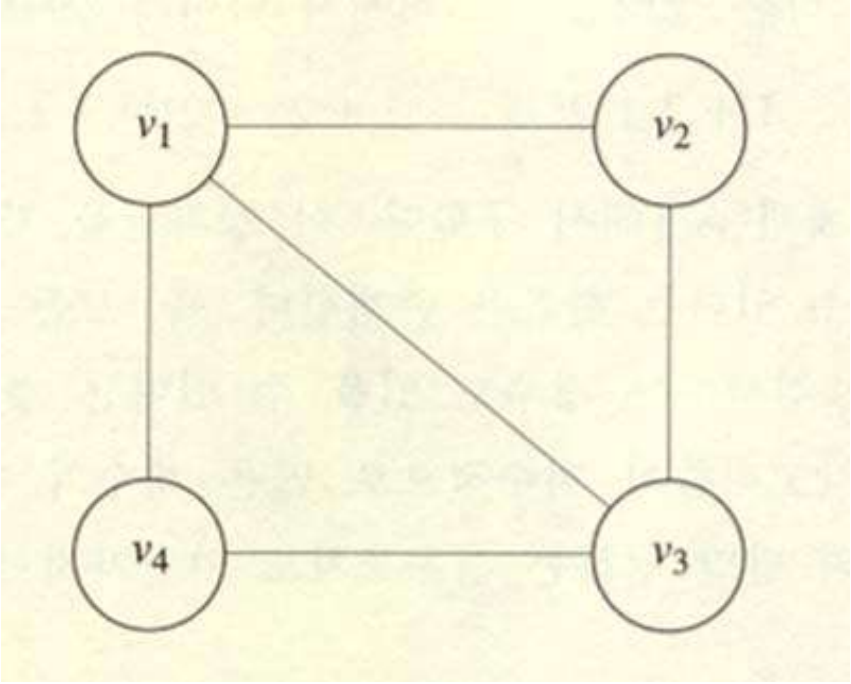
- Monte Carlo 기법은 어떤 입력이 주어졌을 때 점검하게 되는 state space tree의 "전형적인" 경로를 무작위(random)로 생성하여 그 경로 상에 있는 노드의 수를 센다. 이 과정을 여러 번 반복하여 나오는 결과의 평균치를 추정치로 한다.

- 이 기법을 적용하기 위해서 다음 두 조건을 반드시 만족해야 함
  - ✓ state space tree의 같은 level에 있는 모든 노드의 promising 여부를 점검하는 절차는 같아야 한다.
  - ✓ state space tree 의 같은 level 에 있는 모든 노드는 반드시 같은 수의 자식노드를 가지고 있어야 한다.

  ➔ *n*-Queens 문제는 이 두 조건을 만족한다.

2) Graph Coloring (그래프 색칠하기)

Graph Coloring 문제 : m개의 색을 가지고, 인접한 지역이 같은 색이 되지 않도록 지도에 색칠하는 문제



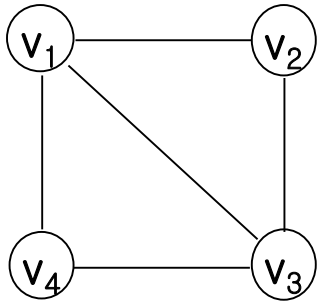
➔ 이 그래프에서 두가지 색으로 문제를 풀기는 불가능하다.

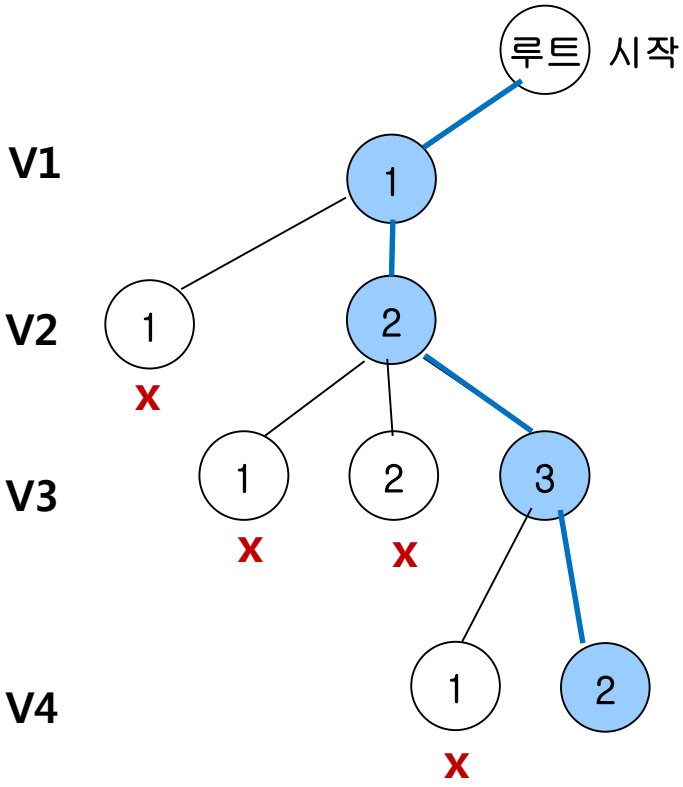
➔세 가지 색 필요

➔세 가지 색으로총 6가지의 해답을 얻을 수 있다.

그래프 색칠하기 : Backtracking 해법

[문] 3개 색으로, 인접한 지역이 같은 색이 되지 않도록 색칠하는 방법을 나타내는 상태 공간 트리 (첫 해답이 형성되는 과정)





➔ v1에는 1번색, v2에는 2번색, v3에는 3번색, v4에는 2번색으로 하면 3가지 색으로 색칠할 수 있음. 즉, Solution Vector = (1,2,3,2).

그래프 색칠하기: 분석

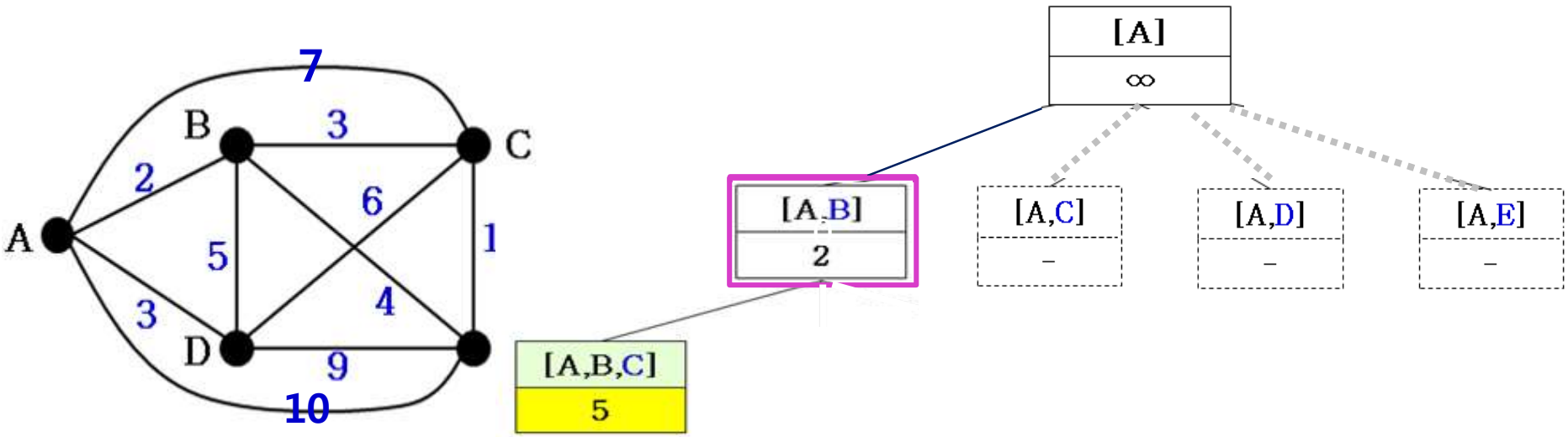
- 색깔 개수: m, 노드 개수: n
- State space tree 상의 노드의 총수 :

$$1 + m + m^2 + \cdots + m^n = \frac{m^{n+1} - 1}{m - 1} \rightarrow m^n$$

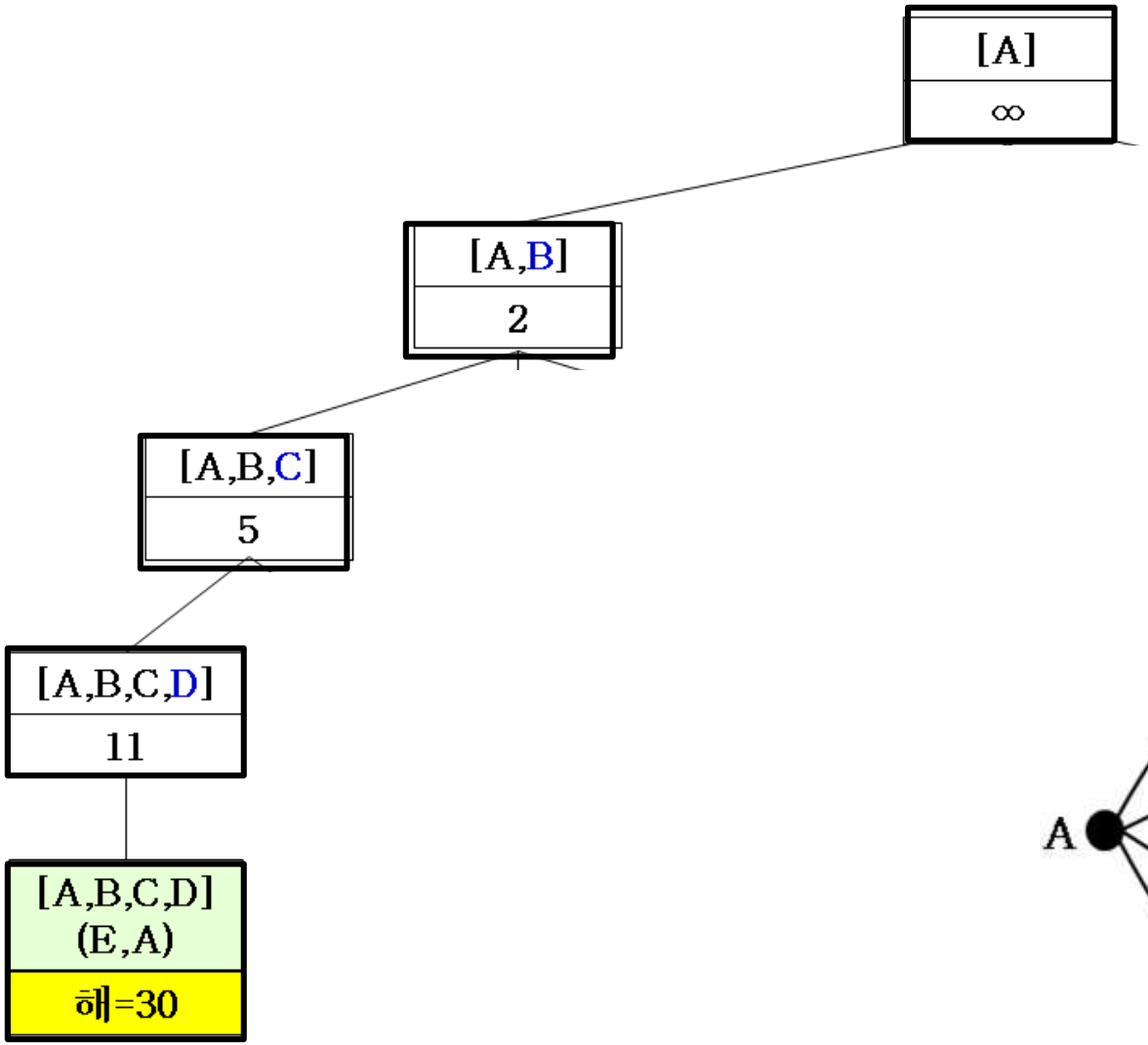
- 여기서도 Monte Carlo 기법을 사용하여 수행시간을 추정할 수 있다.

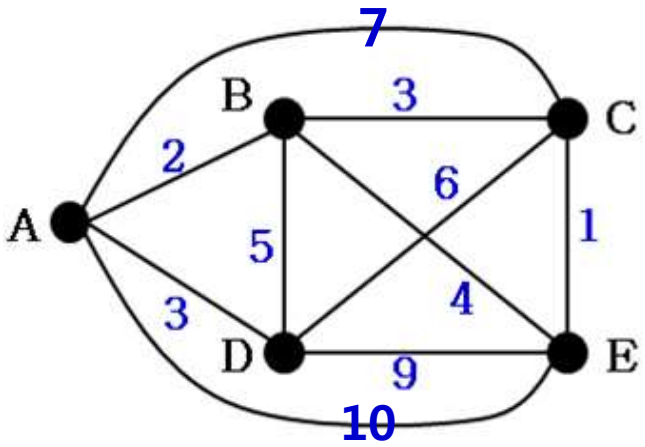
3) 외판원 문제 (TSP)

- 외판원 문제(TSP)를 위한 백트래킹 알고리즘

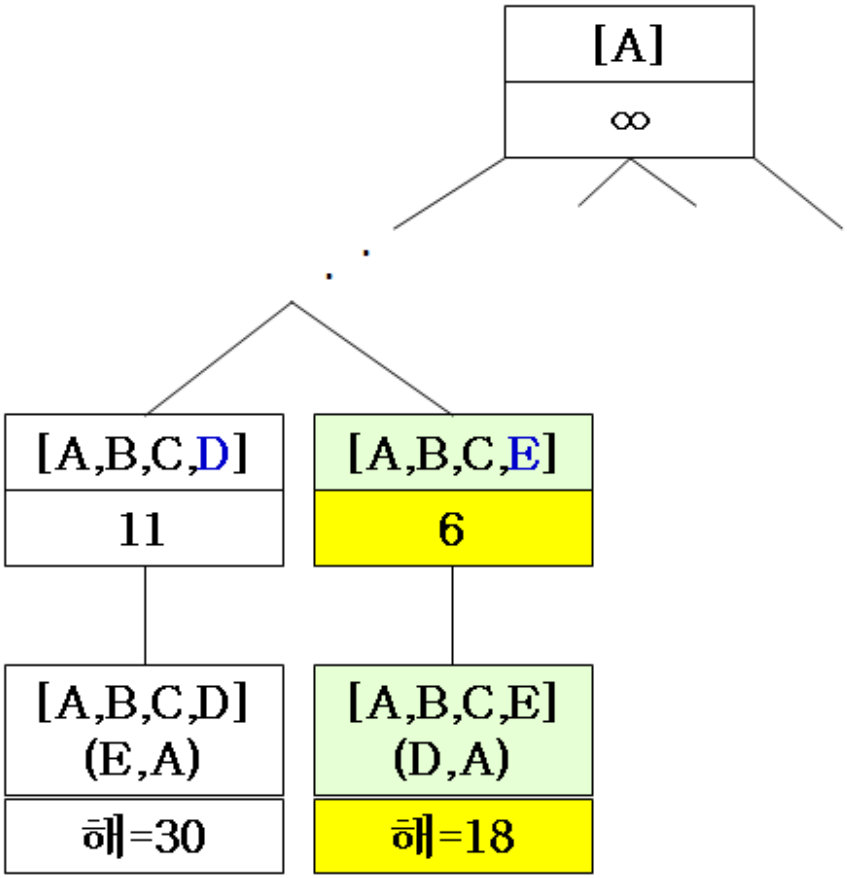


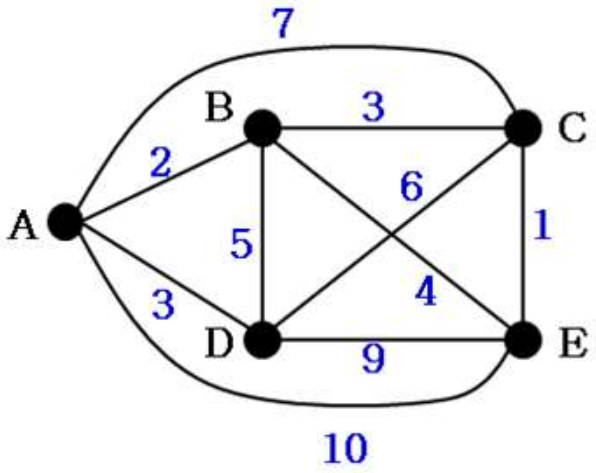
- 이와 같이 계속 탐색이 진행되면 다음과 같이 해를 찾는다. 이때 bestSolution=([A,B,C,D,E,A], 30)이 된다.

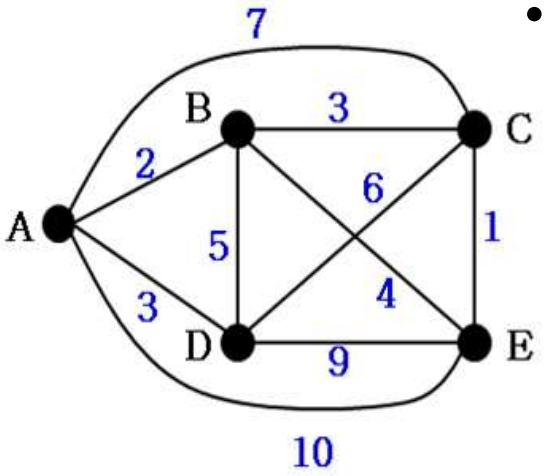




- 첫 번째 해를 찾은 후에는 다음과 같이 수행되며, 이때 더 짧은 해를 찾으므로 bestSolution=([A,B,C,E,D,A], 18)이 된다.

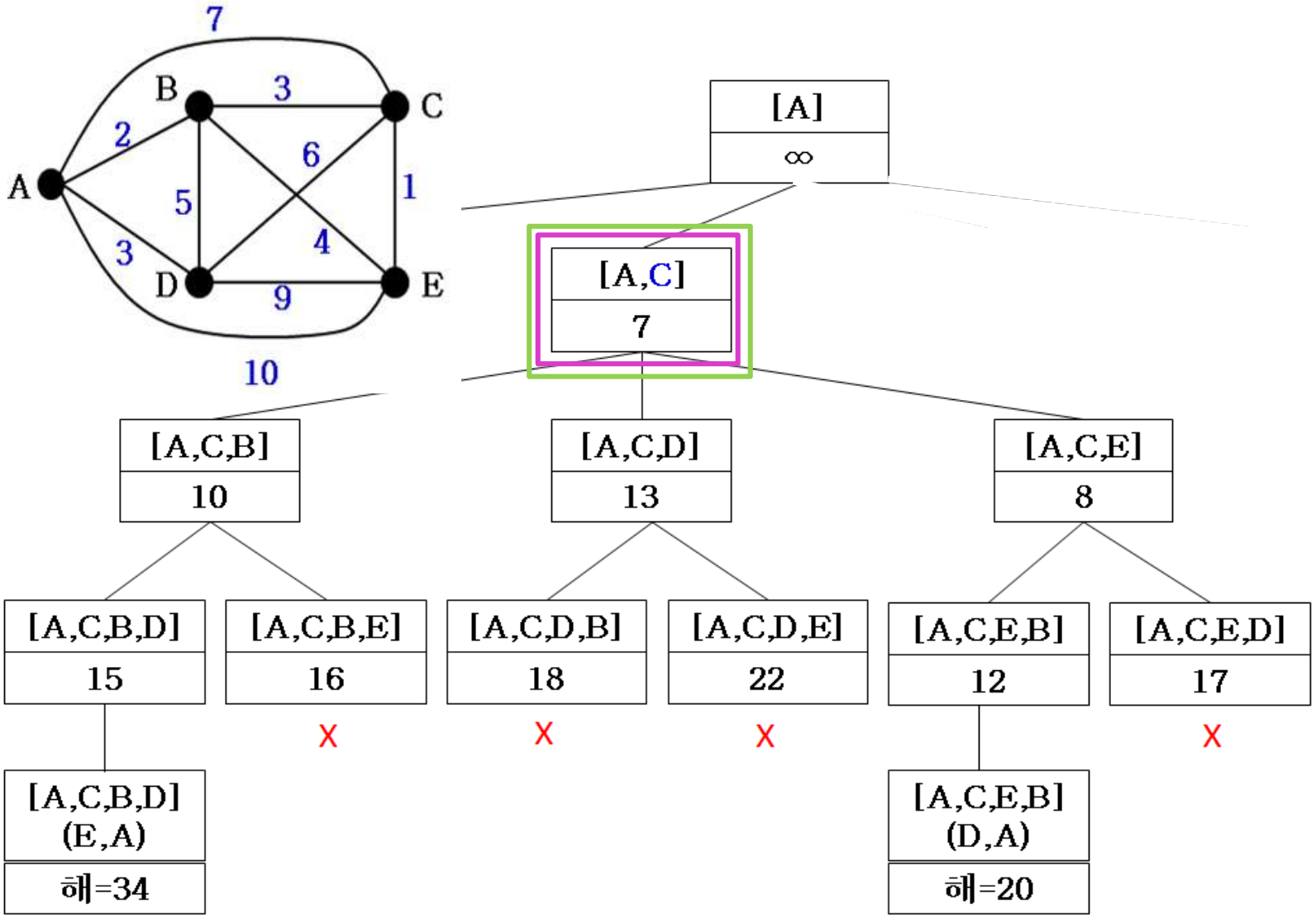




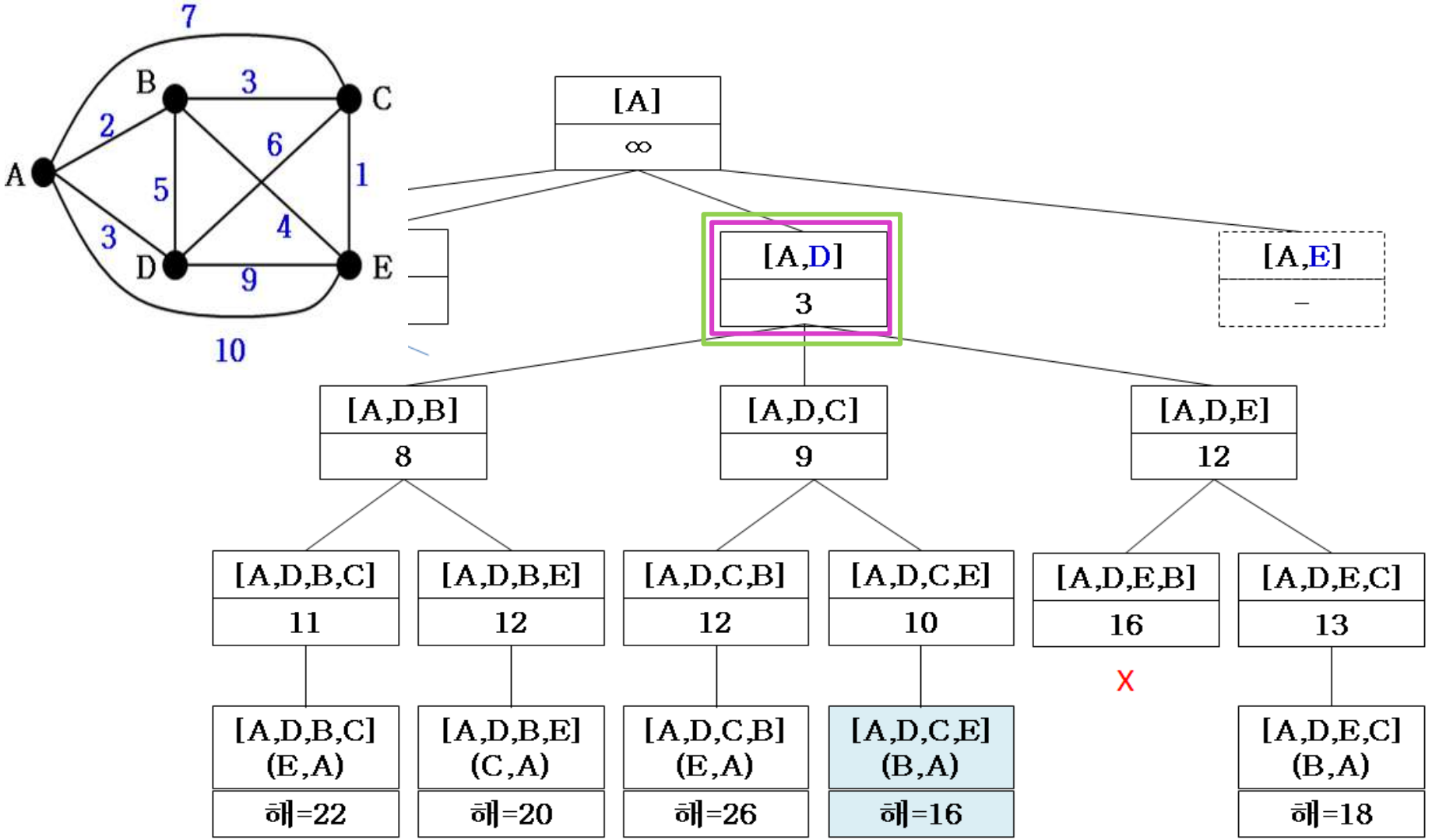


- 다음은 [A,B]에 대해서 모든 수행을 마친 결과로, bestSolution=([A,B,E,C,D,A], 16)이다

- 다음은 tour=[A,C]에 대해서 모든 수행을 마친 결과 bestSolution보다 더 우수한 해는 탐색되지 않았고, x표시된 4개의 상태는 bestSolution의 거리(16)보다 짧지 않아서 가지치기한 것임

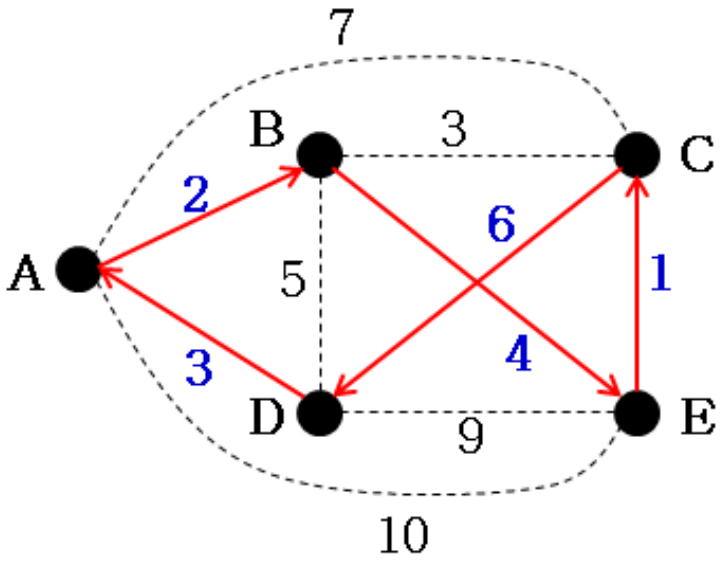


- 다음은 [A,D]에 대해서 모든 수행을 마친 결과 bestSolution보다 더 우수한 해는 탐색되지 않았으나, 같은 거리의 해를 찾는데 이 해는 bestSolution의 역순 즉, [A,B,E,C,D,A]
- x로 표시된 1개 상태는 bestSolution과 같으므로 가지치기



- 마지막으로 tour=[A,E]에 대해서 탐색을 수행하여도 bestSolution보다 더 우수한 해는 발견되지 않음

➔ 따라서 최종해= [A,B,E,C,D,A]이고, 거리=16이다.



5장 끝