
알고리즘분석 실습 자료
10주차
Photo by Piotr Guzik on Unsplash

실습 소요 시간 100분

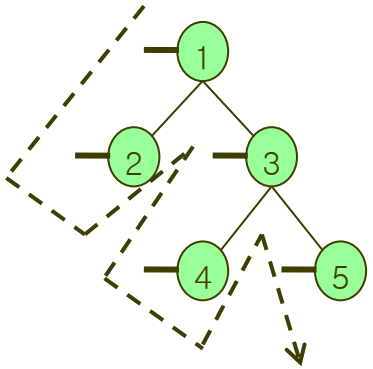
5장 되추적 (Backtracking)

실습프로그램

- ✓ 그래프의 깊이우선검색
- ✓ 그래프의 너비우선검색
- ✓ 5-Queens problem

트리 방문(tree traversal)

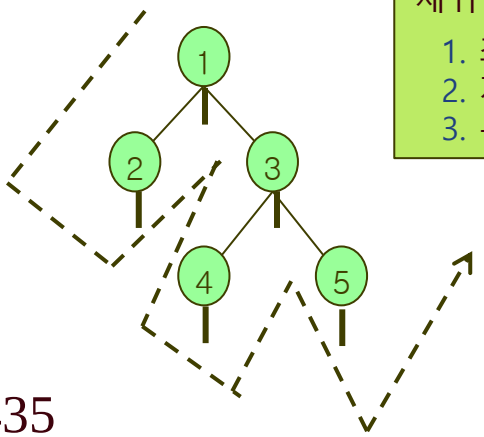
1. preorder



12345

재귀적으로
1. 자신 방문
2. 좌측 방문
3. 우측 방문

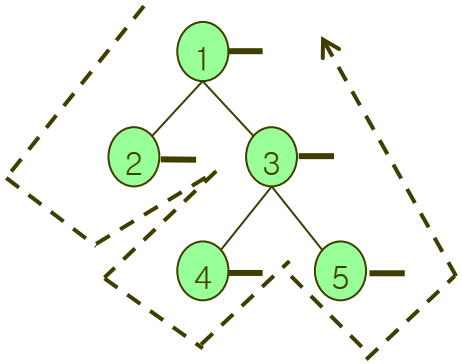
2. inorder



21435

재귀적으로
1. 좌측 방문
2. 자신 방문
3. 우측 방문

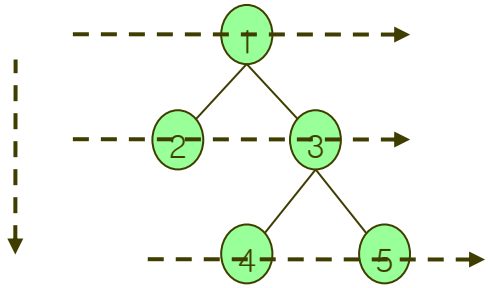
3. postorder



24531

재귀적으로
1. 좌측 방문
2. 우측 방문
3. 자신 방문

4. level order



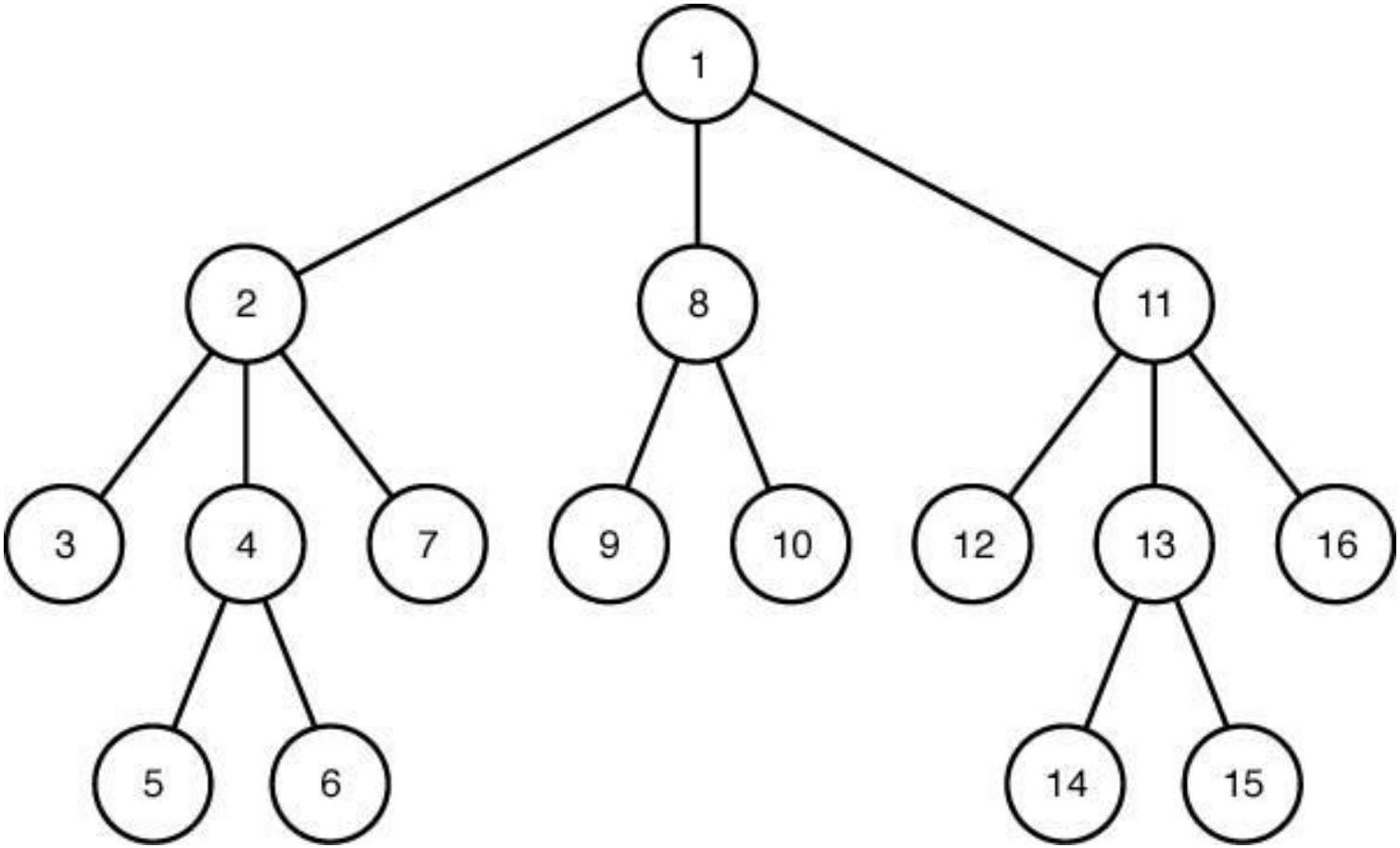
12345

깊이우선검색(depth-first search)

- 뿌리마디(root)가 되는 마디(node)를 먼저 방문한 뒤, 그 마디의 모든 후손마디(descendant)들을 차례로 (보통 왼쪽에서 오른쪽으로) 방문한다.
  (= preorder tree traversal).

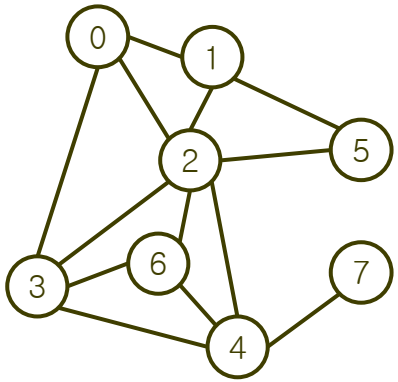
```
void depth_first_tree_search (node v) {
        node u;

        visit v;
        for (each child u of v)
                depth_first_tree_search(u)
}
```

깊이우선검색의 예



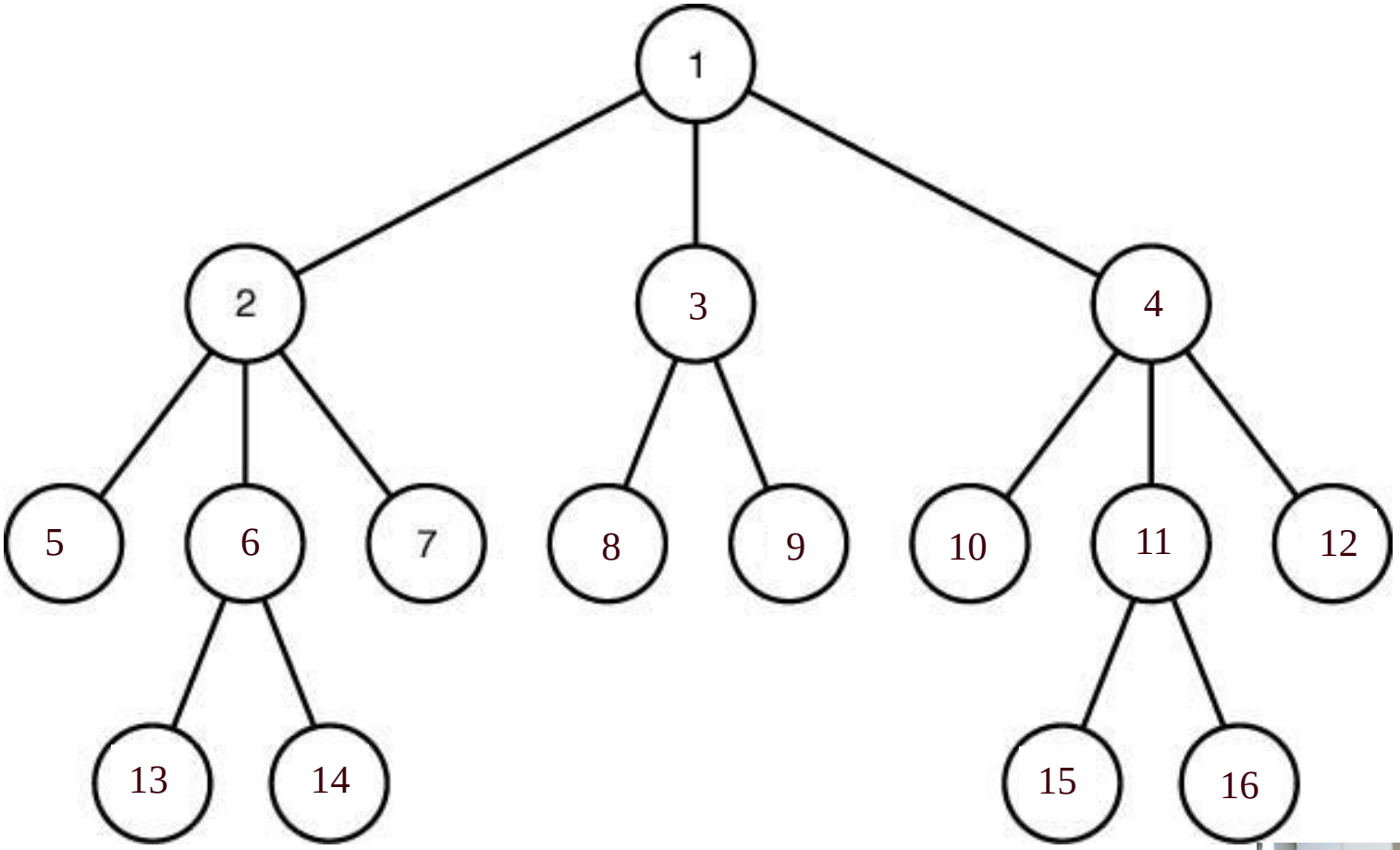
[실습프로그램] 그래프의 깊이우선검색 구현

```
import utility
e={0:[1,2,3], 1:[2,5], 2:[3,4,5,6], 3:[4,6],4:[6,7]}
n=8
a = [ [0 for j in range(0,n)] for i in range(0,n)]
for i in range(0,n-1):
    for j in range(i+1,n):
        if i in e:
           if j in e[i]:
              a[i][j]=1
              a[j][i]=1
utility.printMatrix(a)

visited =n*[0]

def DFS(a,v):


     깊이우선검색 구현


DFS(a,0)
```




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

너비우선검색의 예



python 에서 지원하는 queue class

```
import queue

q=queue.Queue()
q.put('a')
q.put('b')
print(q.qsize())
print(q.get())
print(q.qsize())
print(q.get())
print(q.full())
```

```
2
a
1
b
False
```

```
import queue

q=queue.Queue(3)
q.put('a')
q.put('b')
q.put('c')
if q.full() != True:
    q.put('d')
print(q.qsize())
print(q.get())
print(q.qsize())
print(q.get())
print(q.full())
```

```
3
a
2
b
False
```

q.empty()

```
import queue

q=queue.PriorityQueue()
q.put(3)
q.put(1)
q.put(2)
print(q.qsize())
print(q.get())
print(q.qsize())
print(q.get())
print(q.qsize())
print(q.get())
print(q.empty())
```

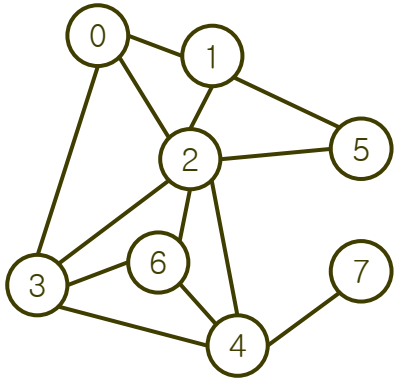
```
3
1
2
2
1
3
True
```

```
import queue

q=queue.LifoQueue()
q.put('a')
q.put('b')
print(q.qsize())
print(q.get())
print(q.qsize())
print(q.get())
```

```
2
b
1
a
```

[실습프로그램] 그래프의 너비우선검색 구현

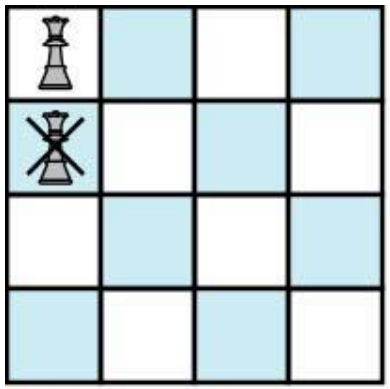
```
import utility
import queue
e={0:[1,2,3], 1:[2,5], 2:[3,4,5,6], 3:[4,6],4:[6,7]}
n=8
a = [ [0 for j in range(0,n)] for i in range(0,n)]
for i in range(0,n-1):
    for j in range(i+1,n):
        if i in e:
           if j in e[i]:
              a[i][j]=1
              a[j][i]=1
utility.printMatrix(a)

visited =n*[0]

def BFS(a,v):
    q=queue.Queue()




    너비우선검색 구현


BFS(a,0)
```



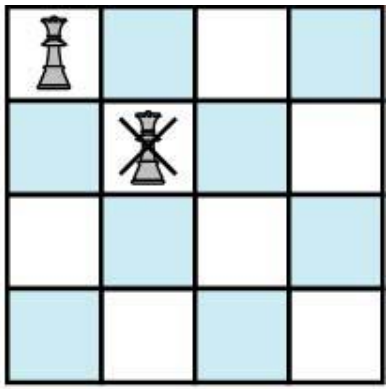

```
>>>
  0   1   1   1   0   0   0   0
  1   0   1   0   0   1   0   0
  1   1   0   1   1   1   1   0
  1   0   1   0   1   0   1   0
  0   0   1   1   0   0   1   1
  0   1   1   0   0   0   0   0
  0   0   1   1   1   0   0   0
  0   0   0   0   1   0   0   0
0
1
2
3
5
4
6
7
```

4-Queens 문제

- 4개의 Queen을 서로 상대방을 위협하지 않도록 4×4 서양장기(chess)판에 위치시키는 문제. 서로 상대방을 위협하지 않기 위해서는 같은 행이나, 같은 열이나, 같은 대각선 상에 위치하지 않아야 한다.

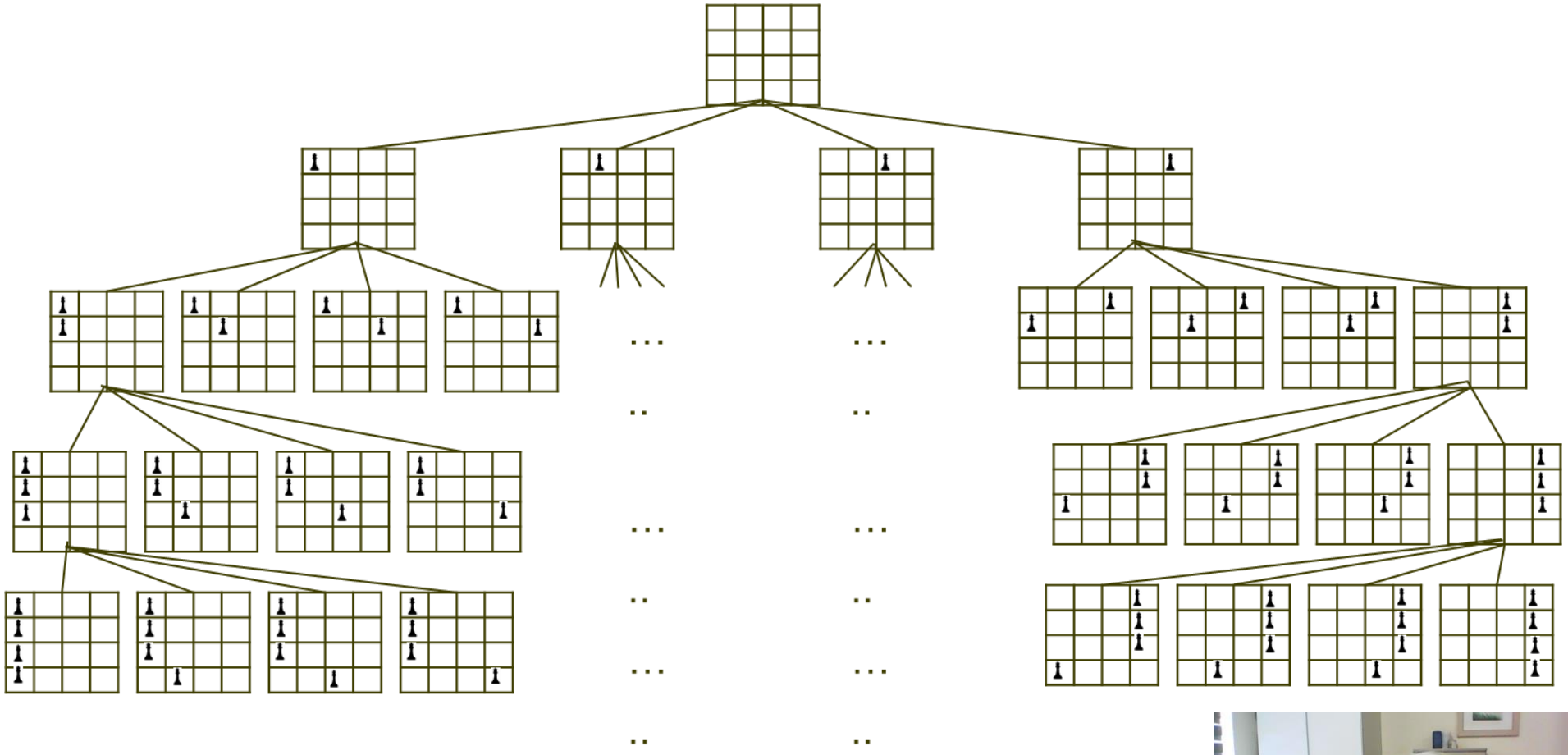


(a)



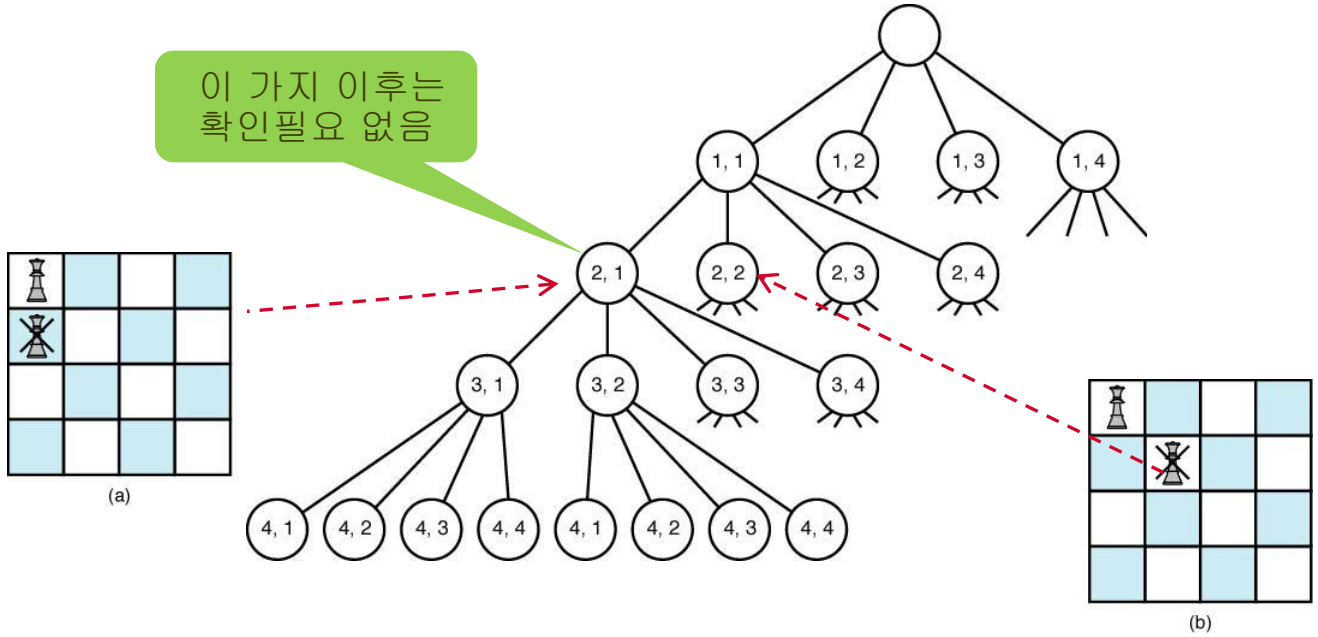
(b)

여왕 배치 상태공간을 계층적으로 표시



상태공간트리(state space tree)

- 뿌리마디에서 잎마디(leaf)까지의 경로는 해답후보(candidate solution)가 되는데, 깊이우선검색을 하여 그 해답후보 중에서 해답을 찾을 수 있다.
- 그러나 이 방법을 사용하면 해답이 될 가능성이 전혀 없는 마디의 후손마디(descendant)들도 모두 검색해야 하므로 비효율적이다.



되추적 기술

- **마디의 유망성:**
  - ✓ 전혀 해답이 나올 가능성이 없는 마디는 유망하지 않다(non-promising)
  - ✓ 그렇지 않으면 유망하다(promising).

- 되추적이란?
  - ✓ 어떤 마디의 유망성을 점검한 후, 유망하지 않다고 판정이 되면 그 마디의 부모마디(parent)로 돌아가서("backtrack") 다음 후손마디에 대한 검색을 계속하게 되는 절차.
  - ✓ 부모마디로 돌아가는 것 → 가지치기(pruning)
  - ✓ 이 과정에서 방문한 마디만으로 구성된 부분트리

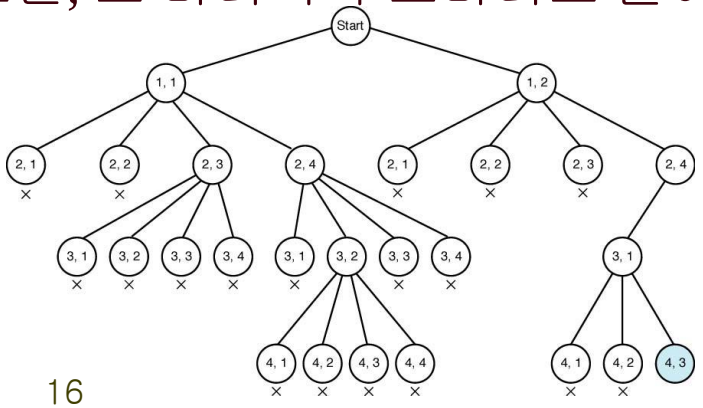
    → 가지친 상태공간 트리(pruned state space tree)

되추적 알고리즘의 개념

- 되추적 알고리즘은 상태공간트리에서 깊이우선검색을 실시하는데,
  - ✓ 유망하지 않은 마디들은 가지쳐서(pruning) 더 이상 하위 노드들을 검색을 하지 않으며,
  - ✓ 유망한 마디에 대해서만 그 마디의 자식마디(children)를 검색한다.

- 이 알고리즘은 다음과 같은 절차로 진행된다.

  1. 상태공간트리의 깊이우선검색을 실시한다.
  2. 각 마디가 유망한지를 점검한다.
  3. 만일 그 마디가 유망하지 않으면, 그 마디의 부모마디로 돌아가서 검색을 계속한다.



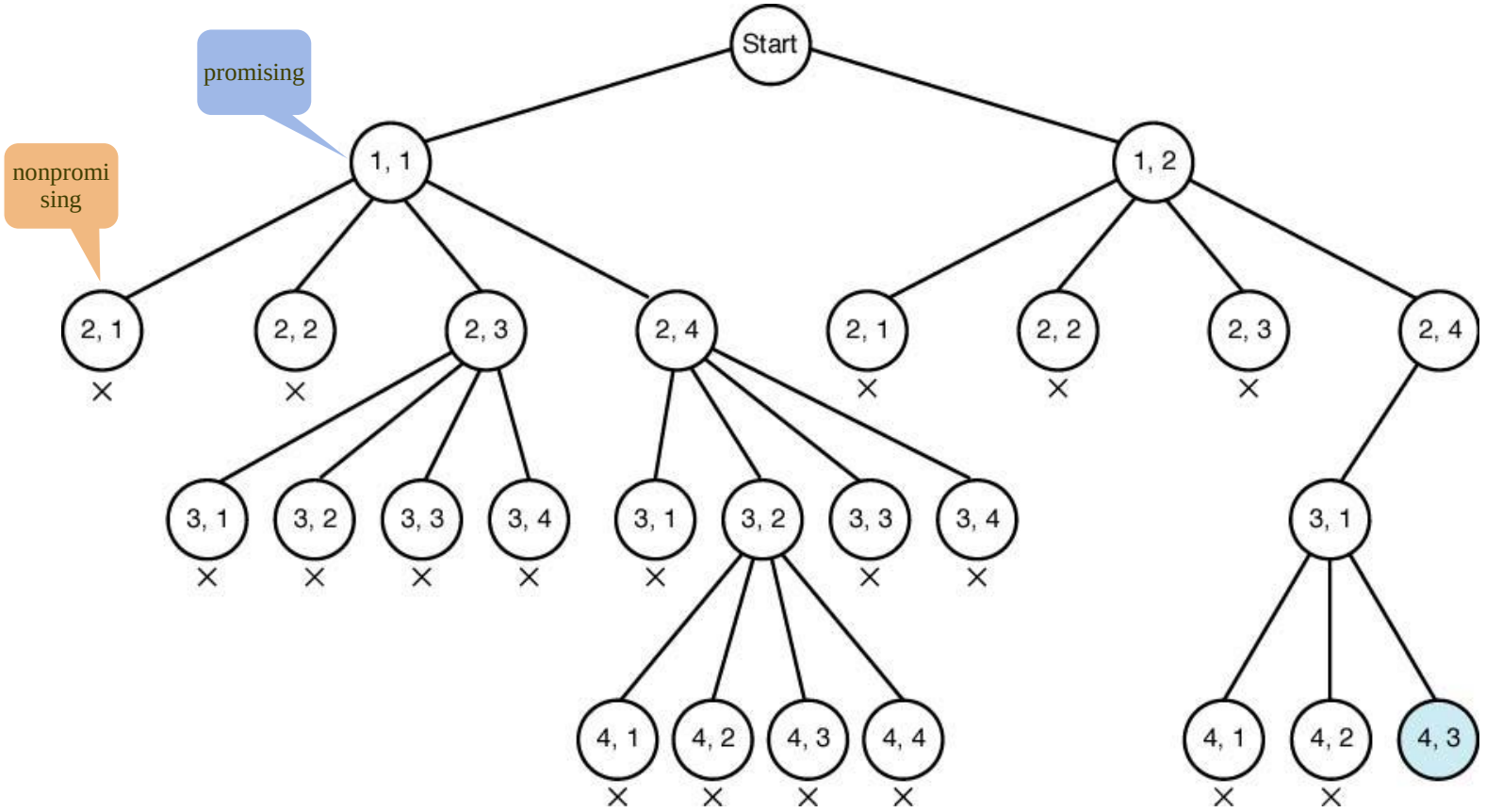
되추적 알고리즘

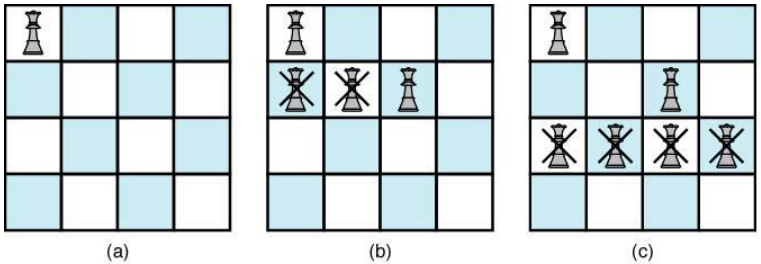
- 일반 되추적 알고리즘:

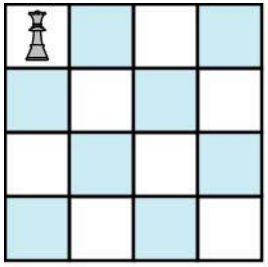
```
void checknode (node v) {
      node u;

  if (promising(v))
      if (there is a solution at v)
          write the solution;
      else
          for (each child u of v)
              checknode(u);
}
```

- 노드를 방문 후 유망성을 검증

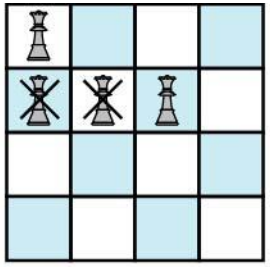
4-Queens 문제의 상태공간트리 (되추적)



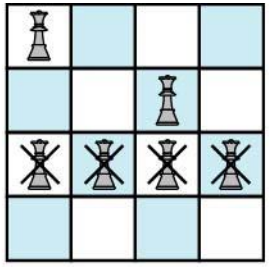




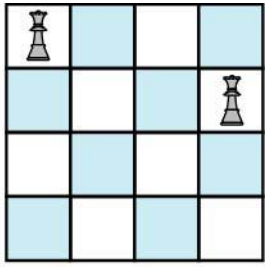
(a)



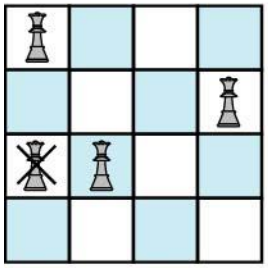
(b)



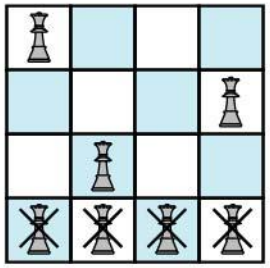
(c)



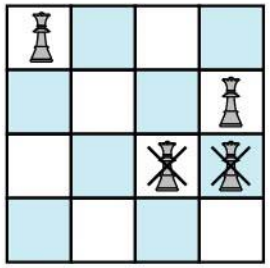
(d)



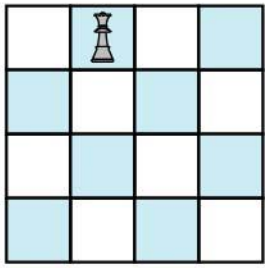
(e)



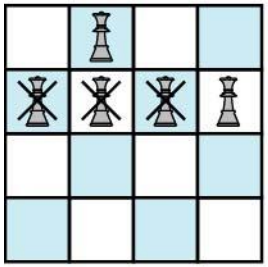
(f)



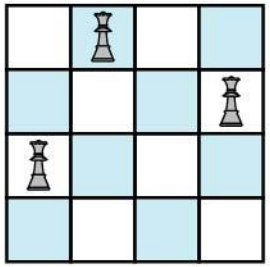
(g)



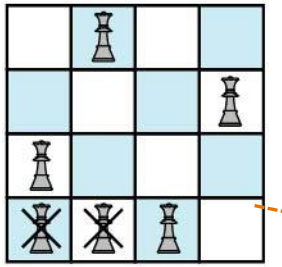
(h)



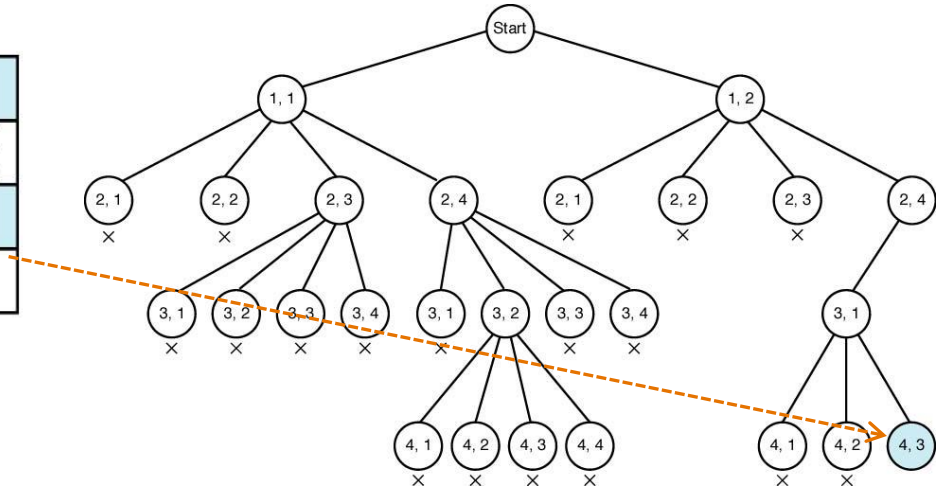
(i)



(j)

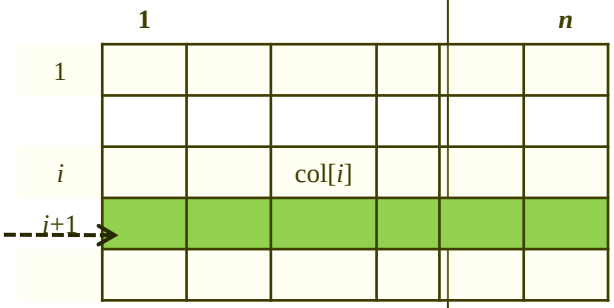


(k)



col[*i*] = *i*번째 queen이 위치한 column 값

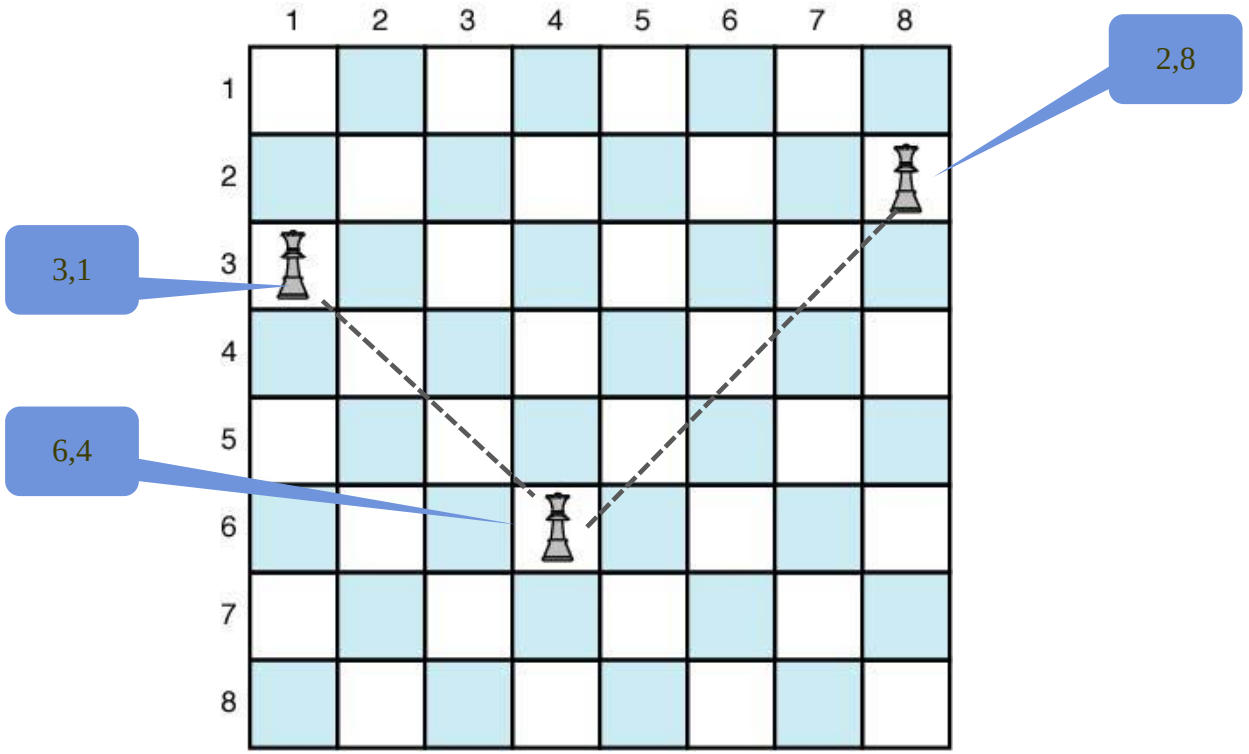
```
void queens(index i){
     index j;
    if(promising(i))
        if (i==n)
            cout << col[1] through col[n];
        else
             for (j=1; j<=n; j++){
                   col[i+1] = j;
                   queens(i+1);
             }
}
bool promising(index i) {
    index k;
    bool switch;

    k=1;
    switch = true;
    while ( k<i  && switch){
          if( col[i]==col[k] || abs(col[i]-col[k]) == i-k)
               switch = false;
          k++;
      }
    return switch;
 }
```



같은 column 인지 확인

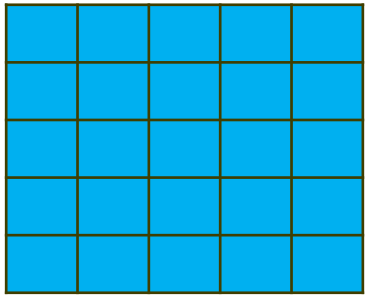
같은 대각에 있는지 확인

Initially, queens(0).


1 2 3 4 5 6 7 8
1
2
3
4
5
6
7
8
2,8
3,1
6,4

[실습프로그램] 5-Queens problem

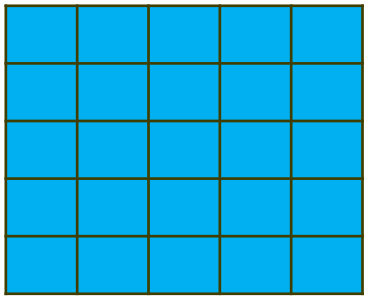
```
def promising(i,col):

     구현


def queens(n,i,col):

     구현



n=5
col=n*[0]
queens(n,-1,col)
```



```
[0, 2, 4, 1, 3]
[0, 3, 1, 4, 2]
[1, 3, 0, 2, 4]
[1, 4, 2, 0, 3]
[2, 0, 3, 1, 4]
[2, 4, 1, 3, 0]
[3, 0, 2, 4, 1]
[3, 1, 4, 2, 0]
[4, 1, 3, 0, 2]
```

[실습프로그램] 5-Queens problem (첫 번째 해까지 찾기)

```
import sys
def promising(i,col):

   구현


def queens(n,i,col):

      구현

n=5
col=n*[0]
queens(n,-1,col)
```



```
[0, 2, 4, 1, 3]
>>>
```