

# Algorithm (알고리즘) - 1

사전 발표 준비 및 전체적인 이론 학습

**알고리즘**이란 무엇이고, 왜 필요할까?

# 알고리즘

Divide and Conquer (분할과 정복)

Dynamic Programming (동적 프로그래밍)

Greedy Algorithm (탐욕 알고리즘)

Backtracking (되추적법)

Branch (분기한정법)

Sort (분류)

Search (탐색)

# 알고리즘

Divide and Conquer (분할과 정복)

Dynamic Programming (동적 프로그래밍)

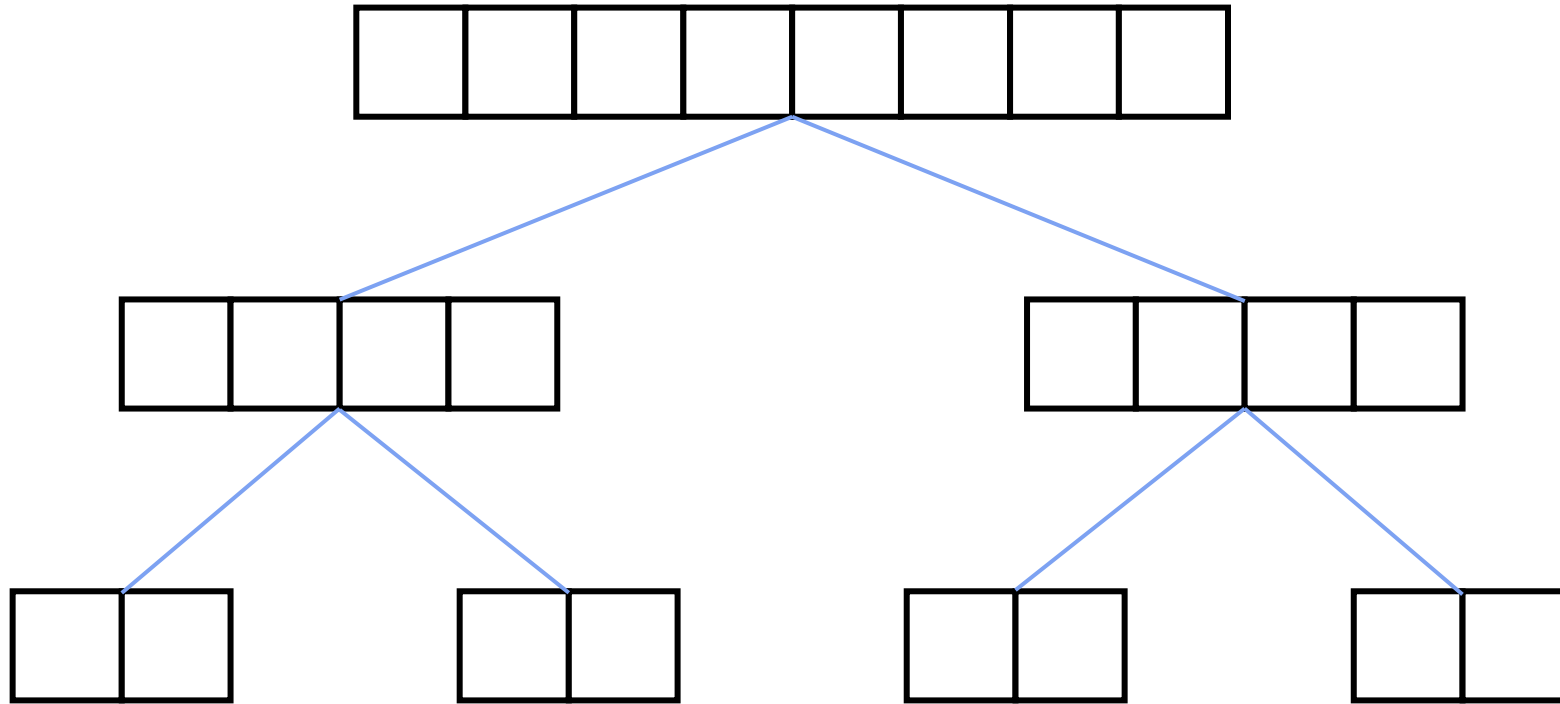
Greedy Algorithm (탐욕 알고리즘)

Backtracking (되추적법)

Branch (분기한정법)

Sort (분류)

Search (탐색)



- 분할된 문제는 기존 문제와 동일하며 input의 size의 크기만 달라진 것
- 분할된 문제는 독립적인 것

→ 구현은 (        )적으로 해결한다.

## Strassen

$$\begin{pmatrix} a1 & a2 \\ a3 & a4 \end{pmatrix} \times \begin{pmatrix} b1 & b2 \\ b3 & b4 \end{pmatrix} = \begin{pmatrix} c1 & c2 \\ c3 & c4 \end{pmatrix}$$

$$C1 = A1*B1 + A2*B3$$

$$C2 = A1*B2 + A2*B4$$

$$C3 = A3*B1 + A4*B3$$

$$C4 = A3*B2 + A4*B4$$

곱셈 : 8회

덧셈 : 4회

$$\begin{pmatrix} C1 & C2 \\ \hline C3 & C4 \end{pmatrix} = \begin{pmatrix} m1 + m4 - m5 + m7 & m3 + m5 \\ m2 + m4 & m1 + m3 - m2 + m6 \end{pmatrix}$$

$$M1 = (A1 + A4) * (B1 + B4)$$

$$M2 = (A3 + A4) * B1$$

$$M3 = A1 * (B2 - B4)$$

$$M4 = A4 * (B3 - B1)$$

$$M5 = (A1 + A2) * B4$$

$$M6 = (A3 - A1) * (B1 + B2)$$

$$M7 = (A2 - A4) * (B3 + B4)$$

곱셈 : 7회

덧셈 : 18회 (뺄셈 포함)

# 알고리즘

Divide and Conquer (분할과 정복)

Dynamic Programming (동적 프로그래밍)

Greedy Algorithm (탐욕 알고리즘)

Backtracking (되추적법)

Branch (분기한정법)

Sort (분류)

Search (탐색)



## 행렬 체인 곱셈 문제(Matrix Chain Multiplication Problem)

1. 행렬의 곱셈은 교환법칙이 성립한다.  $(AB)C = A(BC)$
2. 행렬의 곱셈은 순서에 따라 연산 횟수가 달라진다.

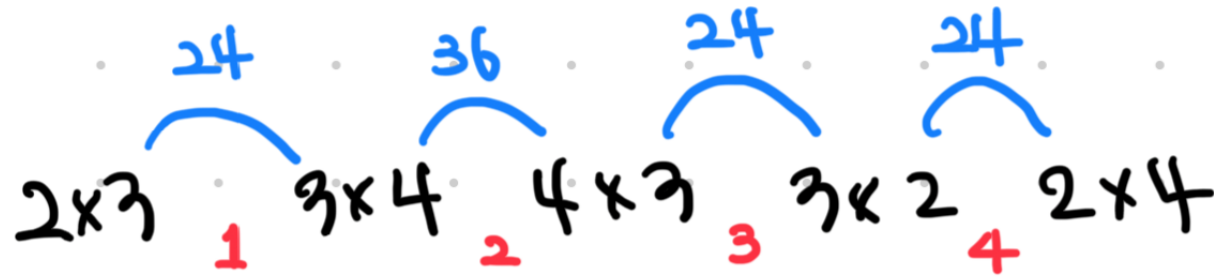
$A(2 \times 5), B(5 \times 4), C(4 \times 3)$

$(AB)C : 2 \times 5 \times 4 + 2 \times 4 \times 3 = 64$

$A(BC) : 2 \times 5 \times 3 + 5 \times 4 \times 3 = 90$

\* 행렬이 커질수록 큰 차이가 난다.

-> 연산 횟수를 최소화하는 곱셈 순서를 찾아라!



< i~j 까지 연산 최솟값 >

j \ i	1	2	3	4	5
1		24			
2			36		
3				24	
4					24
5					

< 최솟값의 분할위치 >

j \ i	1	2	3	4	5
1		1			
2			2		
3				3	
4					4
5					

$$24 + 2 \times 4 \times 3 = 48$$

$$2 \times 3 \quad 3 \times 4 \quad 4 \times 3$$

$$36 + 2 \times 3 \times 3 = 56$$

< i ~ j 까지 연산 최솟값 >

j \ i	1	2	3	4	5
1		24	48		
2			36		

$$24 + 2 \times 4 \times 3 = 48$$

$$2 \times 3 \quad 3 \times 4 \quad 4 \times 3$$

분할 위치 (2)

< 최솟값의 분할위치 >

j \ i	1	2	3	4	5
1		1	2		
2			2		

< i~j 까지 연산 최솟값 >

j \ i	1	2	3	4	5
1		24	48	60	76
2			36	48	72
3				24	56
4					24
5					

< 최솟값의 분할위치 >

j \ i	1	2	3	4	5
1		1	2	1	4
2			2	2	4
3				3	4
4					4
5					

# 〈최소값의 분할위치〉

$j \backslash i$	1	2	3	4	5
1		1	2	1	4
2			2	2	4
3				3	4
4					4
5					

1~5의 최적 분할위치

->  $[1][5] == 4$

-> 1~4, 4로 나누어라!

1~4의 최적 분할위치

->  $[1][4] == 1$

-> 1, 2~4로 나누어라!

2~4의 최적 분할위치

->  $[2][4] == 2$

-> 2, 3~4로 분할

최적 순서

->  $((1(2(3 \times 4)))5)$

# 알고리즘

Divide and Conquer (분할과 정복)

Dynamic Programming (동적 프로그래밍)

Greedy Algorithm (탐욕 알고리즘)

Backtracking (되추적법)

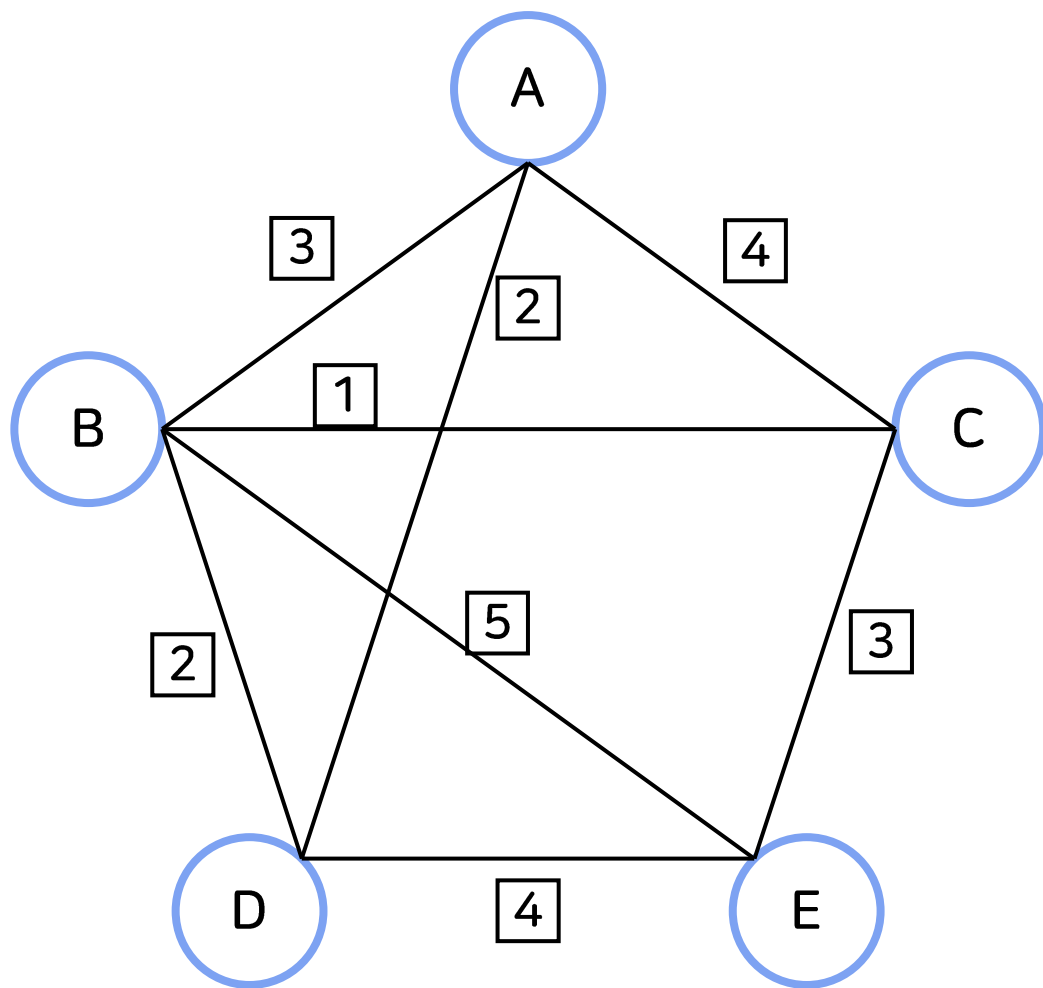
Branch (분기한정법)

Sort (분류)

Search (탐색)

## 최소 신장 트리

N개의 node가 있을 때, cycle없이 모든 node끼리 연결



방법 1

방법 2

# Kruskal : 최소 신장 트리를 찾는 알고리즘

신장 트리? - 하나의 그래프가 있을 때 모든 노드들 간에 서로 연결을 되어있되 사이클이 존재하지 않는 부분 그래프

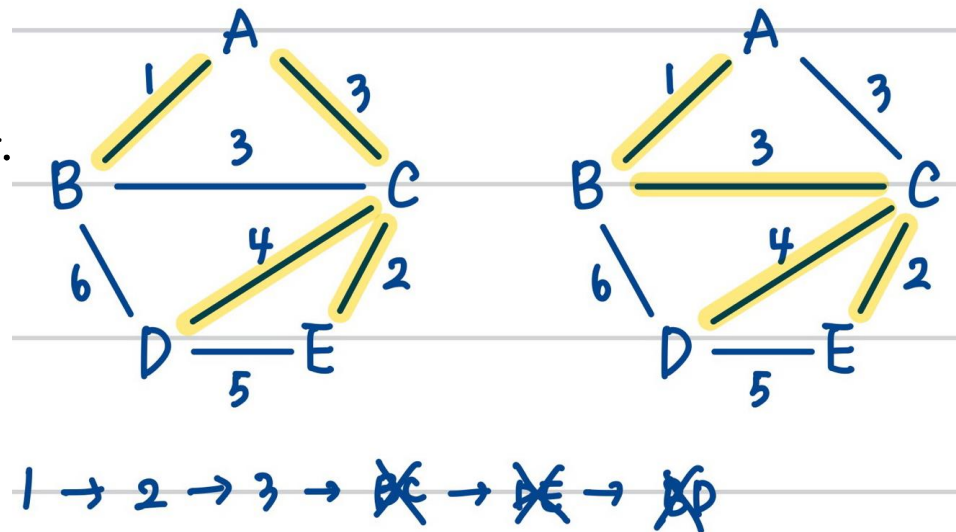
최소? - 간선 비용이 최소

→ 원본 그래프에서 신장 트리를 만드는 경우의 수 중 최소의 간선 비용을 들어서 만들 수 있는 신장 트리

시간복잡도: union-find 알고리즘을 이용하면 간선들을 정렬하는 시간에 달림  $\rightarrow O(E \log E)$

동작 과정

1. 그래프의 간선들을 가중치의 오름차순으로 정렬한다.
2. 정렬된 간선 리스트에서 순서대로 사이클을 형성하지 않는 간선을 선택한다.
  - 즉, 가장 낮은 가중치를 먼저 선택한다.
  - 사이클을 형성하는 간선을 제외한다.
3. 해당 간선을 현재의 MST(최소 비용 신장 트리)의 집합에 추가한다.





# Union - Find

서로소 집합이라고도 함

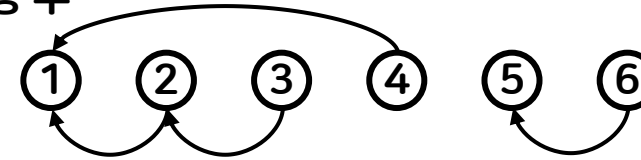
1. 초기단계: 자기 자신을 부모로 가지도록 설정

	①	②	③	④	⑤	⑥
노드 번호	1	2	3	4	5	6
부모 테이블 값	1	2	3	4	5	6

2. Union 연산 수행

Union 연산 정보: (2, 1), (3, 2), (4, 1), (6, 5)

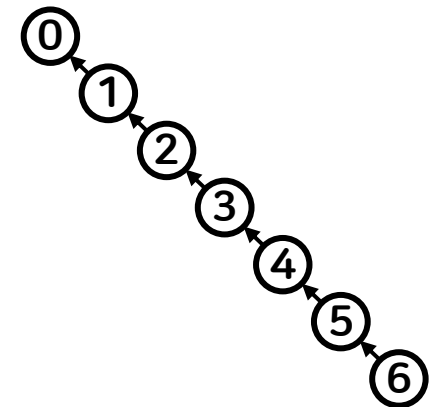
Union 연산 수행 후

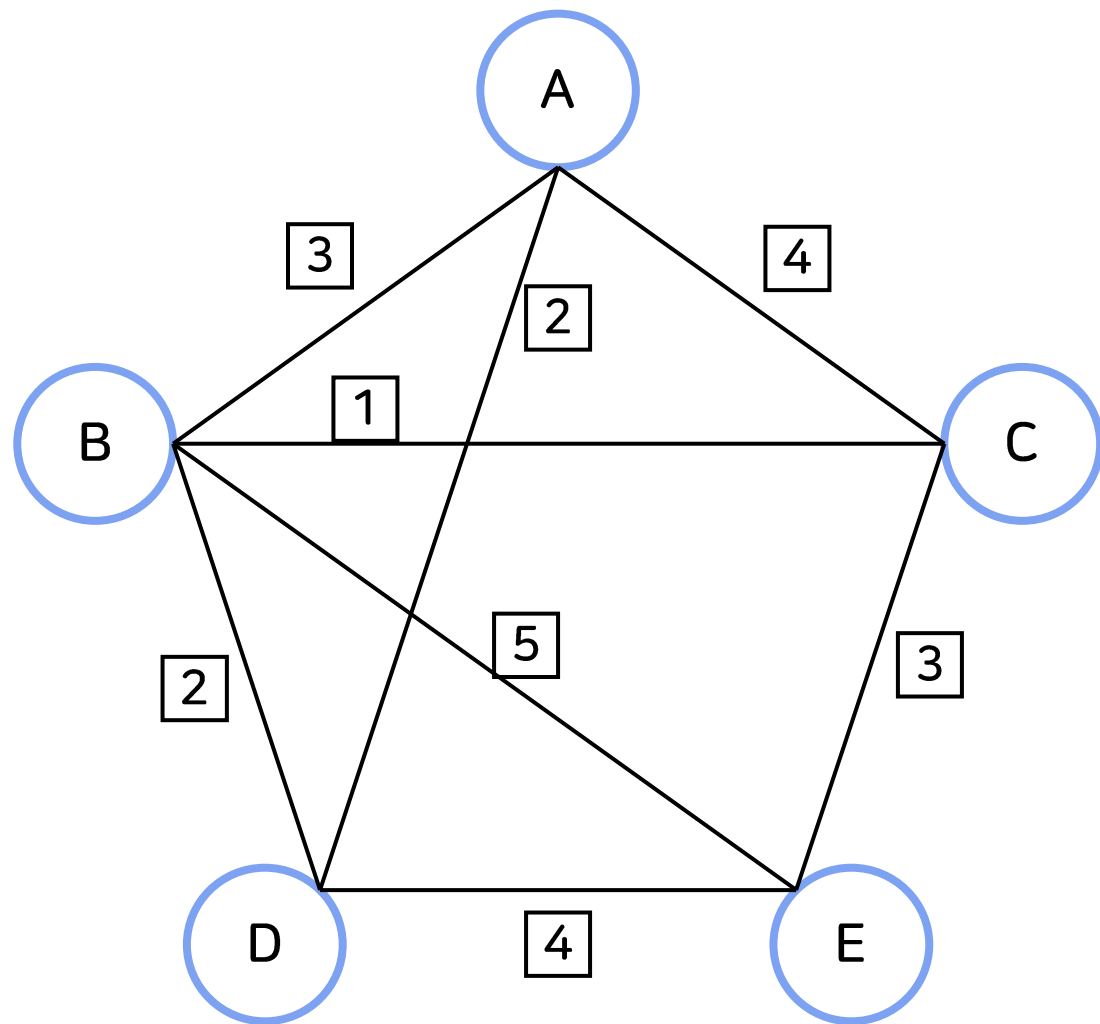


노드 번호	1	2	3	4	5	6
부모 테이블 값	1	1	2	1	5	5

3. Find 연산 수행 - 노드3같은 애들이 재귀적으로 가장 루트인 노드(이 경우엔 노드1)을 찾아가는 과정  
경로 압축 기법으로 최적화 (find 함수를 재귀적으로 호출한 뒤에 부모 테이블 값을 갱신하는 기법)

-> 코드 보기





Prim

Kruskal

# 알고리즘

Divide and Conquer (분할과 정복)

Dynamic Programming (동적 프로그래밍)

Greedy Algorithm (탐욕 알고리즘)

Backtracking (되추적법)

Branch (분기한정법)

Sort (분류)

Search (탐색)

백트래킹은 해를 찾아가는 과정에서 지금의 경로가 해가 될 거 같지 않으면, 해당 경로로 탐색을 더 진행하지 않고 이전 단계로 돌아가 다른 방향으로 탐색을 진행하는 것을 의미한다.

백트래킹은 완전탐색과 유사하지만, 불필요한 탐색을 줄이기 위해 탐색을 중단한다는 점에서 문제를 효과적으로 해결할 수 있다.

### N-Queens 문제

문제 설명:  $n \times n$  체스판에  $n$ 개의 퀸을 서로 충돌하지 않게 놓는 경우의 수를 구하는 문제이다.

사용 알고리즘: dfs, 재귀, 백트래킹

풀이 과정:  $n \times n$ 의 체스판에서 각 퀸은 같은 행, 같은 열, 같은 대각선에 존재하면 안됨

필요한 코드 - 1. 퀸이 해당 구역에 들어가도 문제가 없는지 판단하는 함수(백트래킹에서 불필요한 탐색인지를 감지하는 역할)  
2. 반복문을 돌리면서 퀸을 배치하고 배치 가능한 경우의 수를 카운트 하는 함수

```
✓ def promising (i : int, col : list) :  
    k = 0  
    switch = True  
    while ((k < i) and switch) :  
        if ((col[i] == col[k]) or (abs(col[i] - col[k]) == i - k)) :  
            switch = False  
        k += 1  
    return switch  
|  
✓ def queens (n : int, i : int, col : list, cnt : int, cols : list, nodes : int) :  
    if (promising(i, col)) :  
        if (i == n - 1) :  
            cnt += 1  
            cols.append(col.copy())  
        else :  
            for j in range (n) :  
                nodes += 1  
                col[i + 1] = j  
                cnt, cols, nodes = queens(n, i+1, col, cnt, cols, nodes)  
    return cnt, cols, nodes
```

## Promising 함수

i 번째 행에 퀸을 넣을 때 이전의 퀸의 위치들과 비교하며 가능 여부 판정

## Queens 함수

반복문을 실행시켜 1행의 1열부터 퀸을 배치 시키고 재귀적으로 이후 열에 퀸을 배치시킴, 해당 과정을 1행의 마지막 열까지 퀸을 배치 시킬 때 까지 반복, 최종적으로 퀸을 모두 배치하는 경우의 수를 리턴

```
if (__name__ == "__main__") :
```

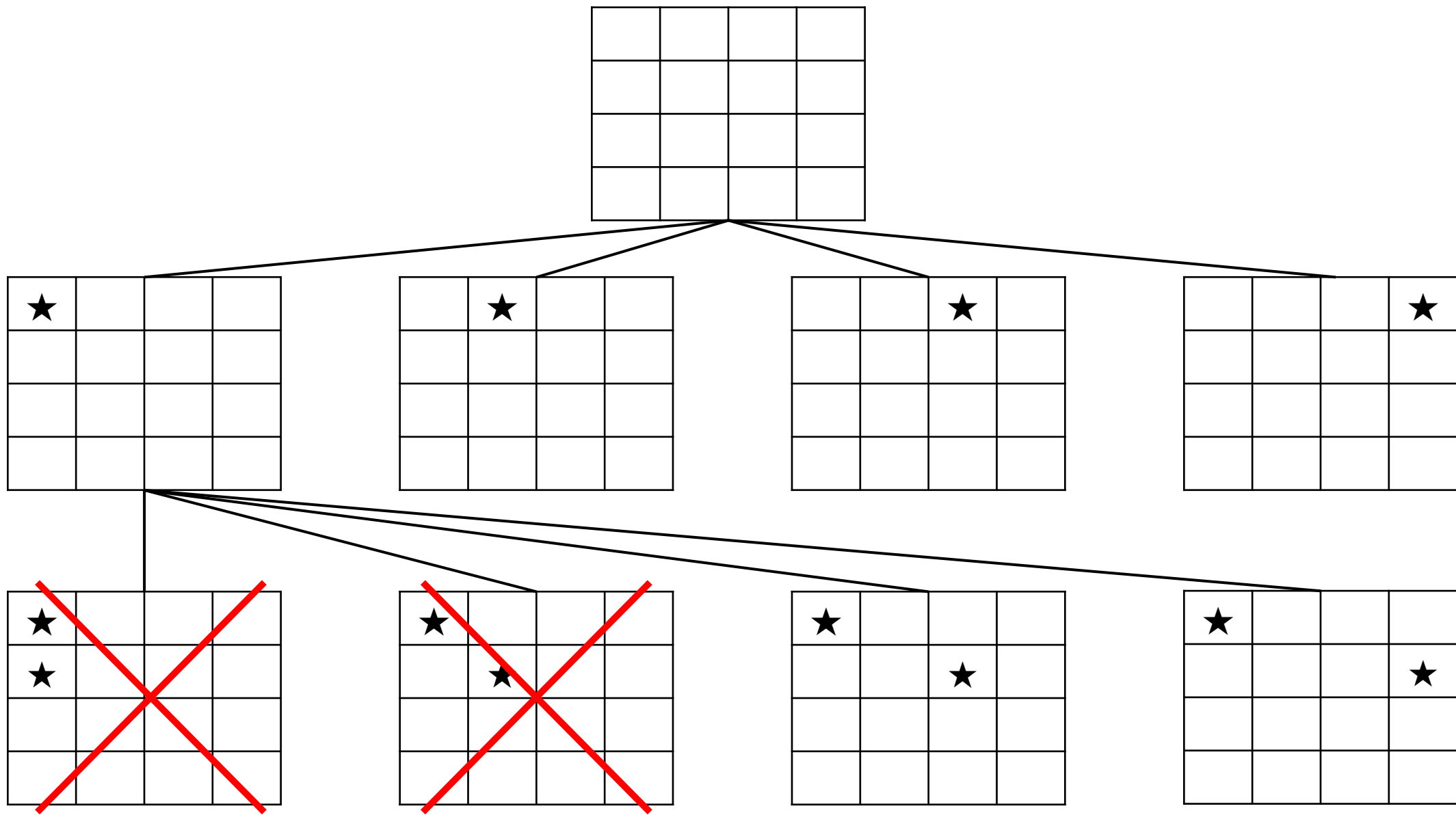
```
    n = 4          # 바둑판의 사이즈. Square
```

```
    cnt, cols, nodes = queens(n, -1, [0] * n, 0, [], 0)
```

```
    print(f" The number of solutions :: {cnt}")          # 해의 갯수  
    print(f" Solution :: {cols}")                        # 경우의 수  
    print(f" The total number of nodes :: {nodes}")     # 상태공간노드의 수
```

## 백트래킹 알고리즘의 이용

Queens 함수에서 탐색 진행 시 promising 함수를 거쳐 불필요한 탐색인지 조사 후 불필요하다면 다음 탐색으로 넘어감





## Task

---

T/F : 지식을 판단하기 위함

서술형 : 알고리즘 동작 과정을 묻기 위함

서술형 : 알고리즘 동작 과정을 묻기 위함 (실제 문제 1개 이상)



# 알고리즘

Divide and Conquer (분할과 정복)

Dynamic Programming (동적 프로그래밍)

Greedy Algorithm (탐욕 알고리즘)

Backtracking (되추적법)

Branch (분기한정법)

Sort (분류)

Search (탐색)

