Algorithm (알고리즘) - 1

사전 발표 준비 및 전체적인 이론 학습

수업 자료의 글꼴 - "나눔스퀘어 라운드[https://hangeul.naver.com/fonts]"
수업 코드 - https://github.com/NET-Organization/24-2-Computer_Science_Study/tree/main/src

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)

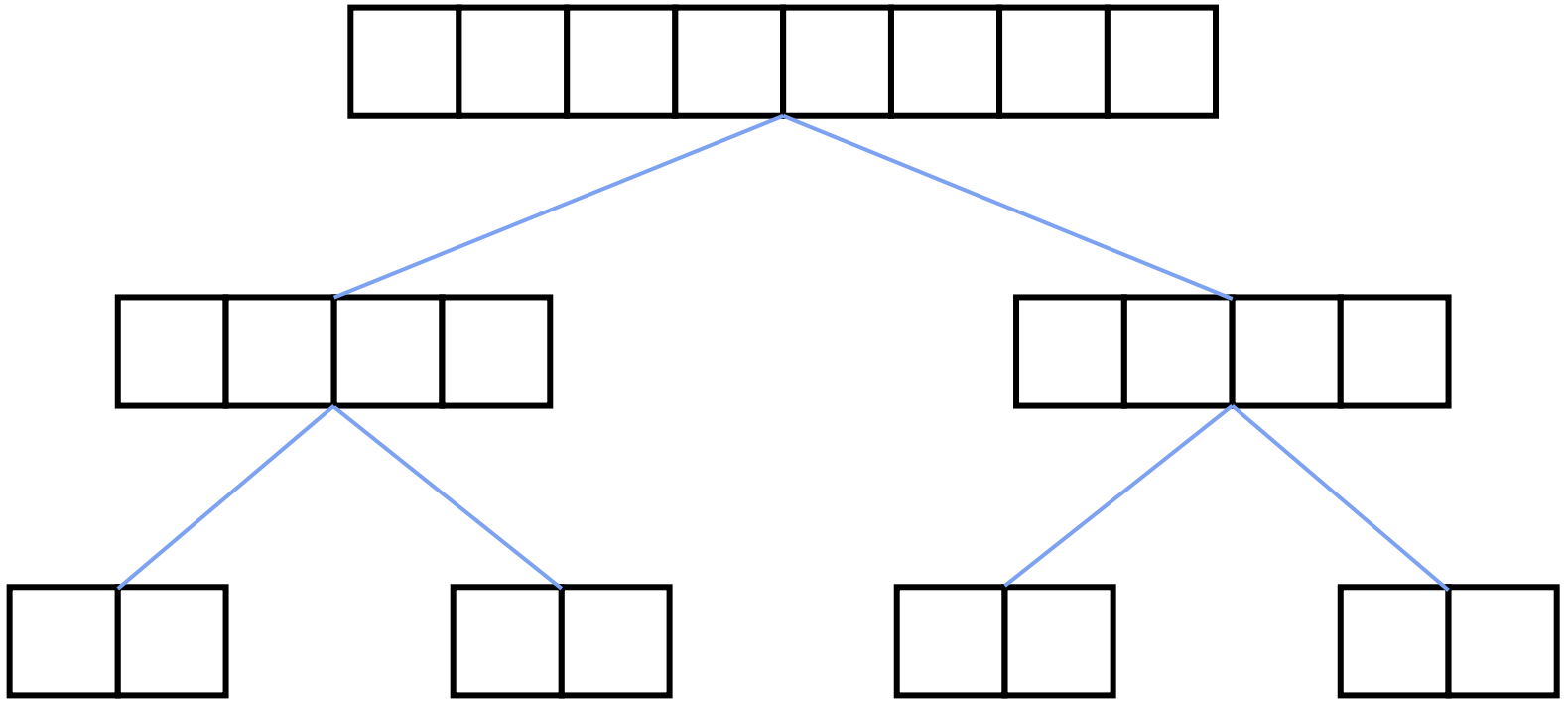
Branch (분기한정법)

Sort (분류)

Search (탐색)

Divide and Conquer (분할과 정복)

양경식



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

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

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.

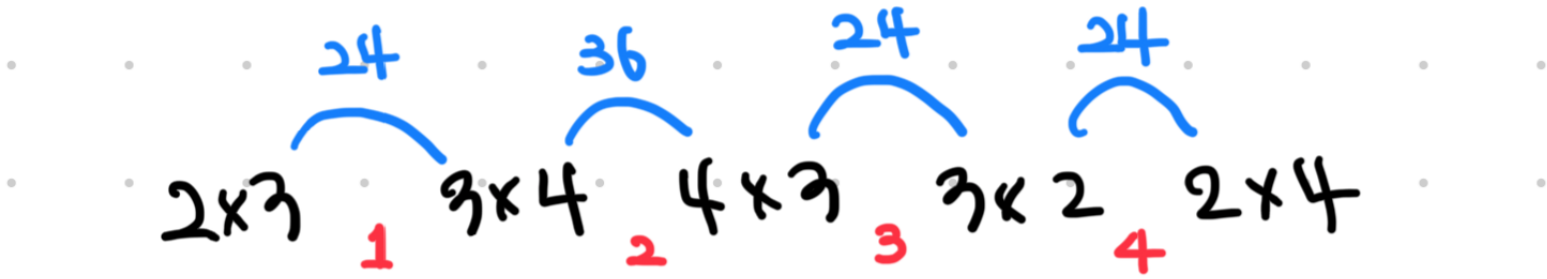
   A(2x5), B(5x4), C(4x3)

   (AB)C : 2x5x4 + 2x4x3 = 64

   A(BC) : 2x5x3 + 5x4x3 = 90

 * 행렬이 커질수록 큰 차이가 난다.

-> 연산 횟수를 최소화하는 곱셈 순서를 찾아라!

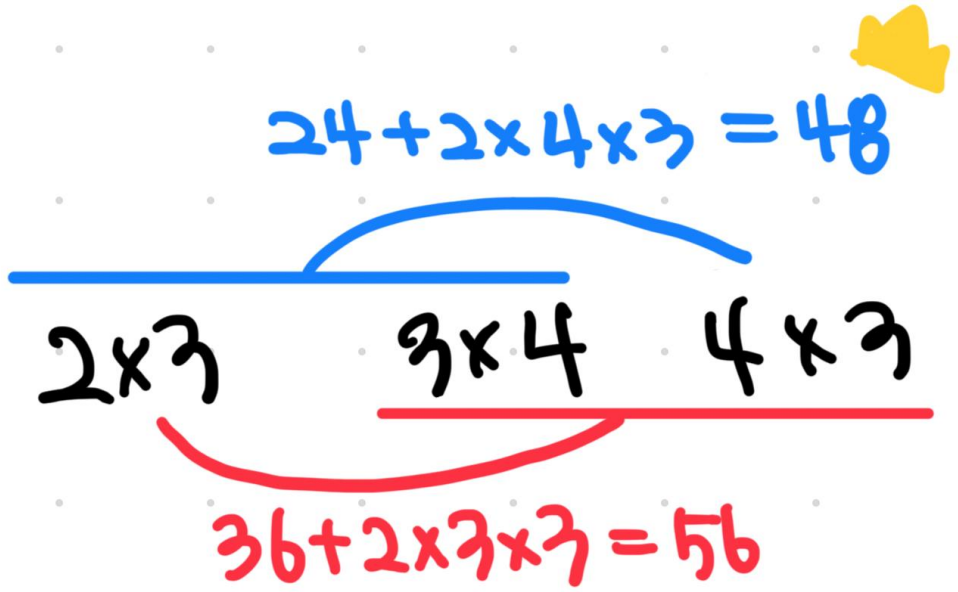


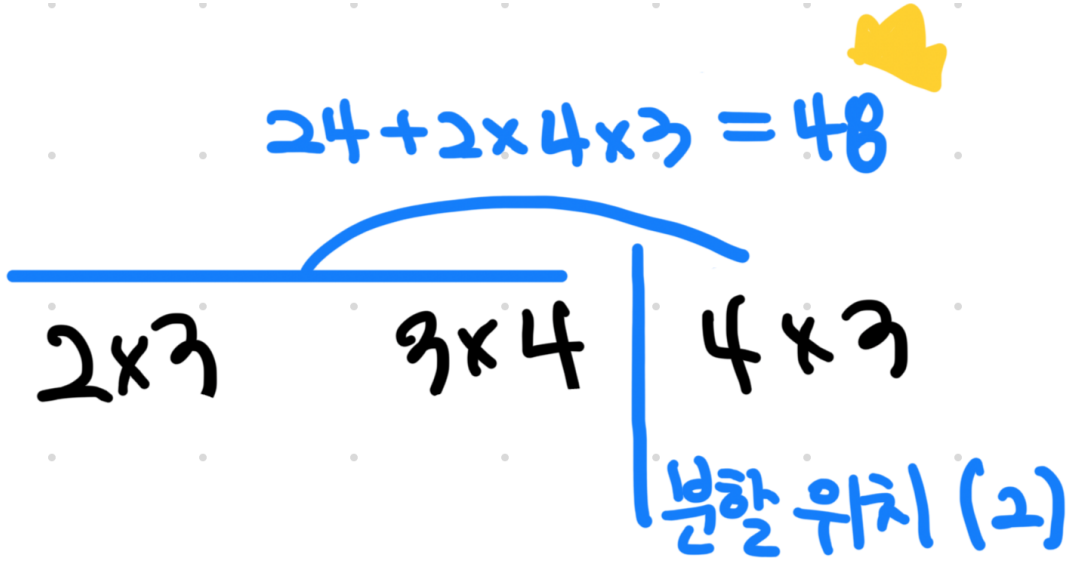
〈i~j 까지 연산 최솟값〉

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					





〈i~j 까지 연산 최솟값〉

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

〈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					

〈최솟값의 분할위치〉

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

1~5 의 최적 분할위치

-> [1][5] == 4

-> 1~4, 4 로 나누어라!

1~4 의 최적 분할위치

-> [1][4] == 1

-> 1, 2~4 로 나누어라!

2~4 의 최적 분할위치

-> [2][4] == 2

-> 2, 3~4 로 분할

최적 순서

-> ((1 (2 (3 x 4))) 5)

알고리즘

Divide and Conquer (분할과 정복)

Dynamic Programming (동적 프로그래밍)

Greedy Algorithm (탐욕 알고리즘)

Backtracking (되추적법)

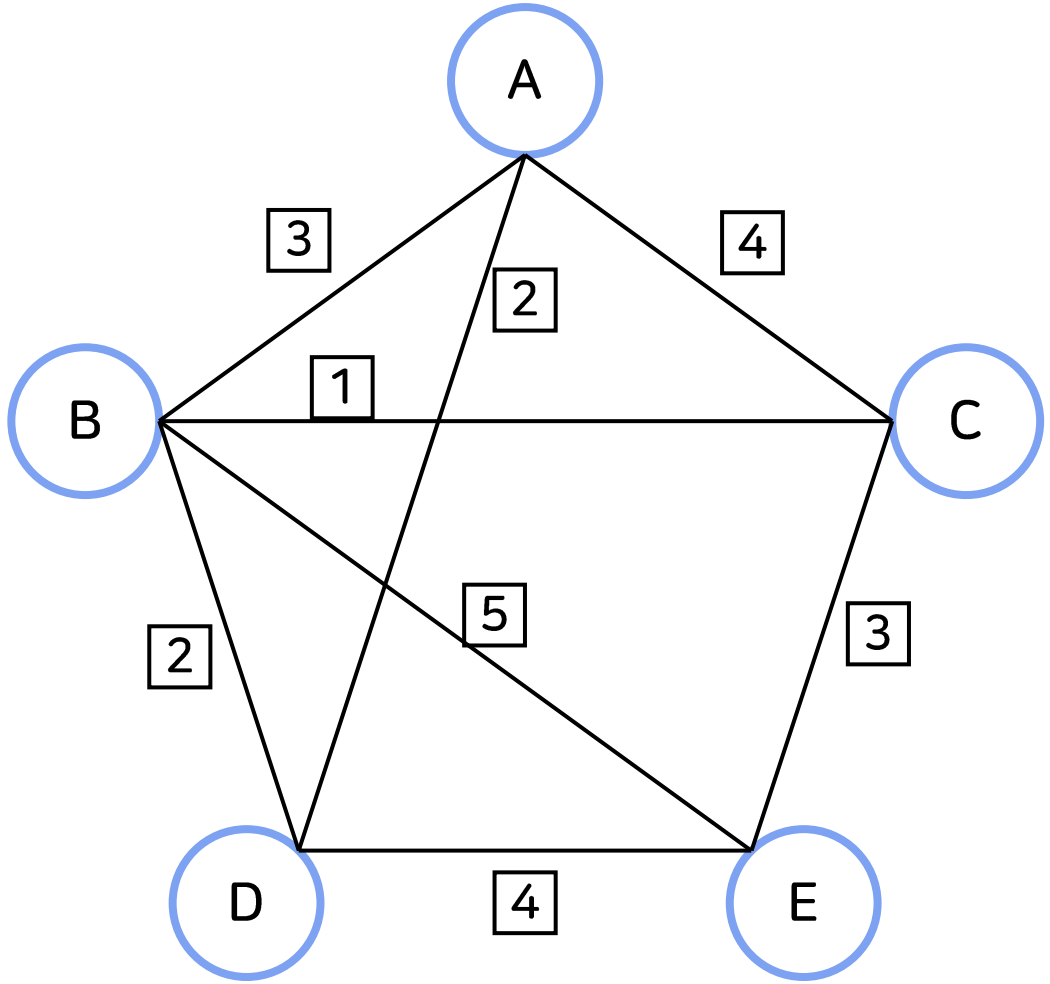
Branch (분기한정법)

Sort (분류)

Search (탐색)

최소 신장 트리

N개의 node가 있을 때, cycle없이 모든 node끼리 연결



방법 1

방법 2

Greedy Algorithm (탐욕 알고리즘)

김혜란

:: 최적의 값을 구해야 하는 상황에서 사용되는 근시안적인 방법론으로 '각 단계에서 최적이라고 생각되는 것을 선택' 하며 최종적인 해답에 도달하는 알고리즘

:: 주로 문제를 분할 가능한 문제들로 분할한 뒤, 각 문제들에 대한 최적해를 구한 뒤 이를 결합하여 전체 문제의 최적해를 구하는 경우에 주로 사용

- 성립 조건

1. 탐욕적 선택 속성(Greedy Choice Property) : 앞의 선택이 이후의 선택에 영향을 주지 않는다.

2. 최적 부분 구조 조건(optimal substructure) : 문제에 대한 최종 해결 방법은 부분 문제에 대한 최적 문제 해결 방법으로 구성된다

- 단계

1. 선택 절차(Selection Procedure): 현재 상태에서의 최적의 해답을 선택한다.

2. 적절성 검사(Feasibility Check): 선택된 해가 문제의 조건을 만족하는지 검사한다.

3. 해답 검사(Solution Check): 원래의 문제가 해결되었는지 검사하고, 해결되지 않았다면 선택 절차로 돌아가 위의 과정을 반복한다.

Kruskal : 최소 신장 트리를 찾는 알고리즘

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

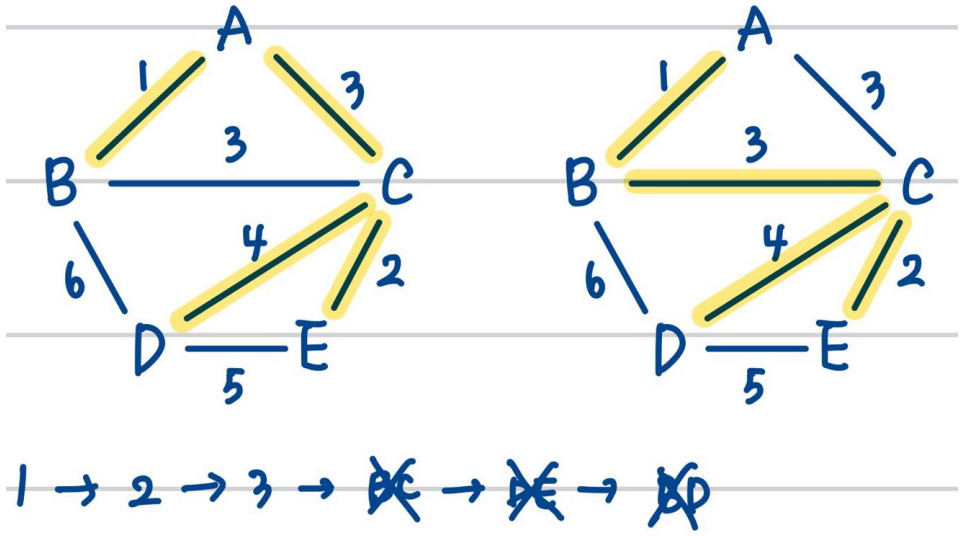
최소? - 간선 비용이 최소

➔원본 그래프에서 신장 트리를 만드는 경우의 수 중 최소의 간선 비용을 들여서 만들 수 있는 신장 트리

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

동작 과정

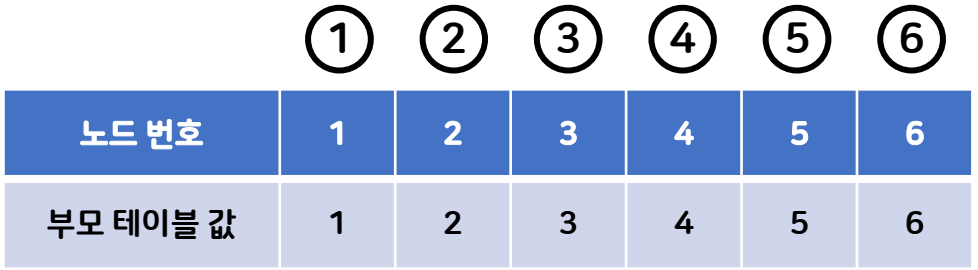
1. 그래프의 간선들을 가중치의 오름차순으로 정렬한다.
2. 정렬된 간선 리스트에서 순서대로 사이클을 형성하지 않는 간선을 선택한다.
- 즉, 가장 낮은 가중치를 먼저 선택한다.
- 사이클을 형성하는 간선을 제외한다.
3. 해당 간선을 현재의 MST(최소 비용 신장 트리)의 집합에 추가한다.



Union - Find

서로소 집합이라고도 함

1. 초기단계: 자기 자신을 부모로 가지도록 설정

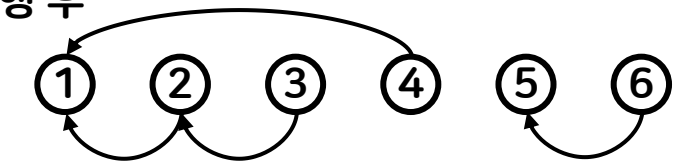


노드 번호	1	2	3	4	5	6
부모 테이블 값	1	2	3	4	5	6

2. Union 연산 수행

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

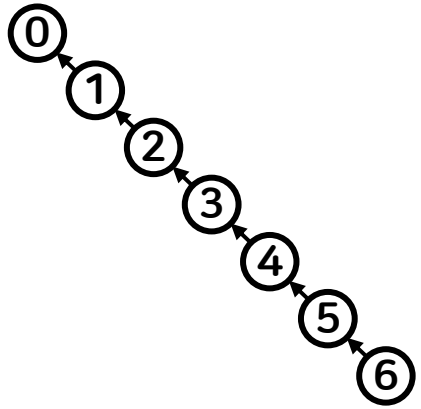
Union 연산 수행 후



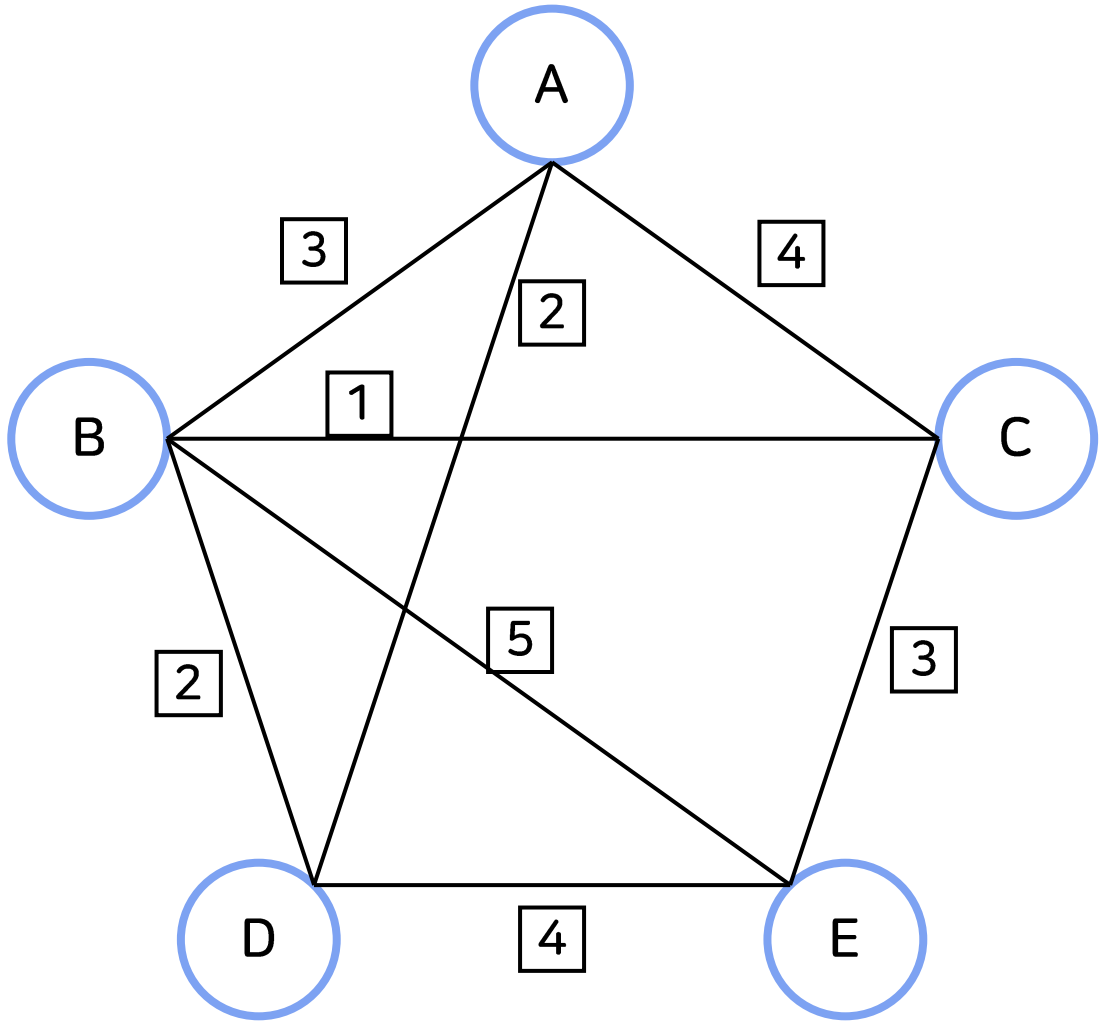
노드 번호	1	2	3	4	5	6
부모 테이블 값	1	1	2	1	5	5

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)

Backtracking (되추적법)

문강민

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: nxn 체스판에 n개의 퀸을 서로 충돌하지 않게 놓는 경우의 수를 구하는 문제이다.
사용 알고리즘: dfs, 재귀, 백트래킹

풀이 과정: nxn의 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

Backtracking (되추적법)

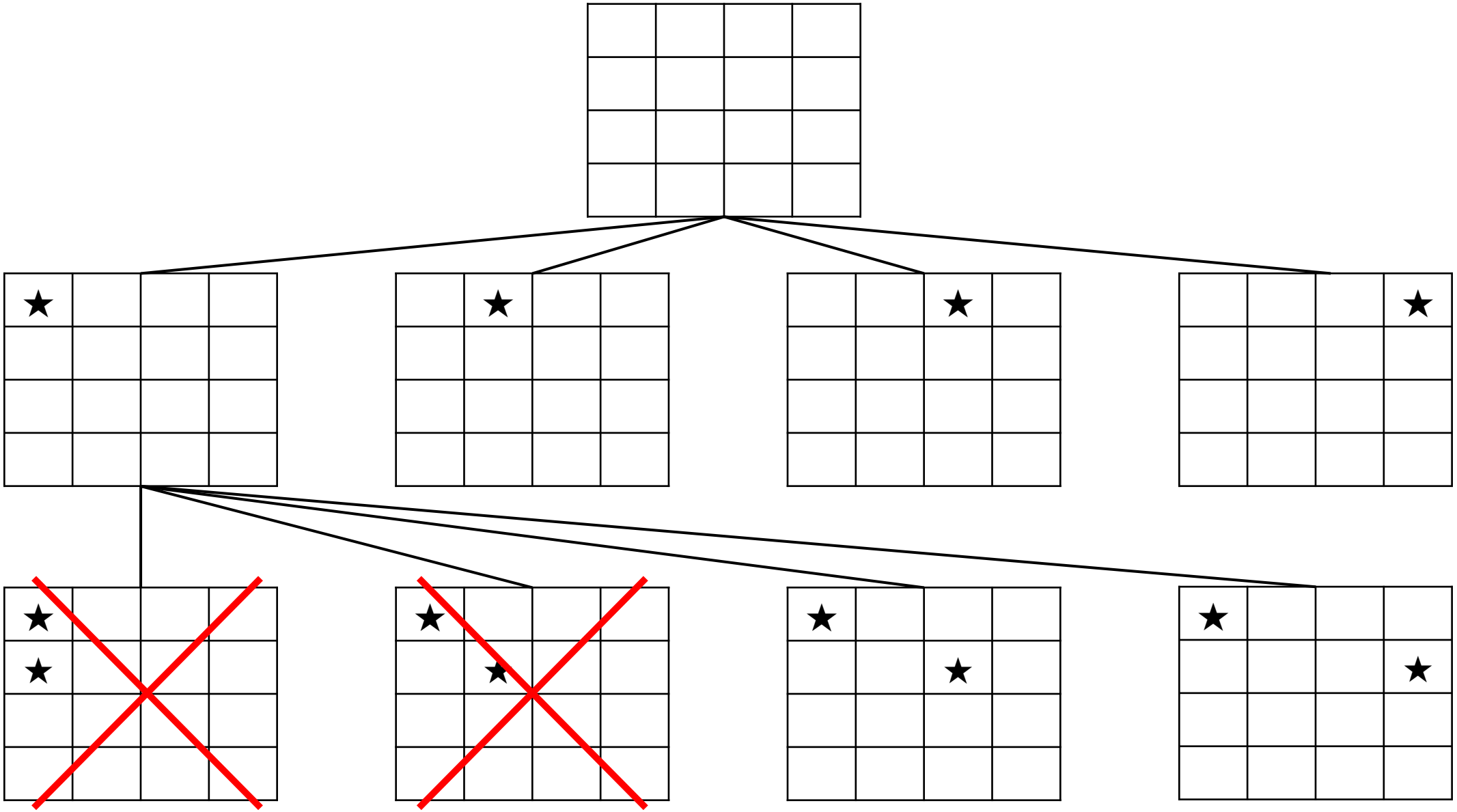
문강민

```python
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

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
```

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

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)