# 25-1 이니로
# 알고리즘 멘토링

멘토 - 김수성

# 음악프로그램 / 2623

## 백준 2623 / https://www.acmicpc.net/problem/2623

- 1 4 3
- 6 2 5 4
- 2 3

남일이가 할 일은 이 순서들을 모아서 전체 가수의 순서를 정하는 것이다. 남일이는 잠시 생각을 하더니 6 2 1 5 4 3으로 순서를 정했다. 이렇게 가수 순서를 정하면 세 보조 PD가 정해온 순서를 모두 만족한다. 물론, 1 6 2 5 4 3으로 전체 순서를 정해도 괜찮다.

경우에 따라서 남일이가 모두를 만족하는 순서를 정하는 것이 불가능할 수도 있다. 예를 들어, 세 번째 보조 PD가 순서를 2 3 대신에 3 2로 정해오면 남일이가 전체 순서를 정하는 것이 불가능하다. 이번에 남일이는 우리 나라의 월드컵 4강 진출 기념 음악제의 PD를 맡게 되었는데, 출연 가수가 아주 많다. 이제 여러분이 해야 할 일은 보조 PD들이 가져온 순서들을 보고 남일이가 가수 출연 순서를 정할 수 있도록 도와 주는 일이다.

보조 PD들이 만든 순서들이 입력으로 주어질 때, 전체 가수의 순서를 정하는 프로그램을 작성하시오.

### 입력

첫째 줄에는 가수의 수 N과 보조 PD의 수 M이 주어진다. 가수는 번호 1, 2,...,N 으로 표시한다. 둘째 줄부터 각 보조 PD가 정한 순서들이 한 줄에 하나씩 나온다. 각 줄의 맨 앞에는 보조 PD가 담당한 가수의 수가 나오고, 그 뒤로는 그 가수들의 순서가 나온다. N은 1이상 1,000이하의 정수이고, M은 1이상 100이하의 정수이다.

### 출력

출력은 N 개의 줄로 이뤄지며, 한 줄에 하나의 번호를 출력한 다. 이들은 남일이가 정한 가수들의 출연 순서를 나타낸다. 답이 여럿일 경우에는 아무거나 하나를 출력 한다. 만약 남일이가 순서를 정하는 것이 불가능할 경우에는 첫째 줄에 0을 출력한다.

# 음악프로그램 / 2623

**각 입력에 대해서 같은 줄에 있는 인원끼리는 순서를 지킬 때 될 수 있는 순서 중 하나를 출력 하는 문제**

- 143
- 6254
- 23

1 -> 4 -> 3　　2 -> 3

6 -> 2 -> 5 -> 4

예제 입력 1 복사

```
6 3
3 1 4 3
4 6 2 5 4
2 2 3
```

예제 출력 1 복사

```
6
2
1
5
4
3
```

# 음악프로그램 / 2623

1 -> 4 -> 3　　2 -> 3

6 -> 2 -> 5 -> 4

같은 줄에 있는 입력끼리 간선을 만들어주면 됨

- 1 4 3
- 6 2 5 4
- 2 3

예제 입력 1 복사

```
6 3
3 1 4 3
4 6 2 5 4
2 2 3
```

예제 출력 1 복사

```
6
2
1
5
4
3
```

# 음악프로그램 / 2623

## 또한 위상 정렬이 불가능 할 때 0을 출력

순서를 정하는 것이 불가능할 경우에는 첫째 줄에 0을 출력한다.

## 위상 정렬이 불가능한 조건 -> 사이클 존재

## 사이클이 존재하면 위상 정렬이 끝났을 때 indegree가 0이 아닌 어떤 인덱스 i가 존재함

# 음악프로그램 / 2623

C++

```cpp
const ll MAX = 1010;
const ll INF = 1e12;
ll n, m, ind[MAX], a[MAX];
vector <ll> adj[MAX], result;
queue <ll> q;
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m;
    while(m--){
        ll sz; cin >> sz;
        for(int i = 1;i <= sz;i++) cin >> a[i];
        for(int i = 1;i < sz;i++){
            // a[i] -> a[i + 1] 인 간선
            adj[a[i]].push_back(a[i + 1]);
            ind[a[i + 1]]++;
        }
    }

    for(int i = 1;i <= n;i++){
        if(!ind[i]) q.push(i);
    }

    // 위상정렬
    while(!q.empty()){
        ll cur = q.front(); q.pop();
        result.push_back(cur);
        for(auto& nxt : adj[cur]){
            if(!--ind[nxt]) q.push(nxt);
        }
    }

    bool flag = 1;
    for(int i = 1;i <= n;i++){
        // indegree가 0이 아닌 값이 있으면
        // 사이클이 존재 -> 위상 정렬 불가
        if(ind[i]) flag = 0;
    }

    if(flag) for(auto& i : result) cout << i << "\n";
    else cout << 0;

    return 0;
}
```

# 음악프로그램 / 2623

## Python

```python
MAX = 1010
INF = 10**12

n, m = map(int, input().split())
ind = [0] * MAX
a = [0] * MAX
adj = [[] for _ in range(MAX)]
result = []
q = deque()
```

```python
for _ in range(m):
    data = list(map(int, input().split()))
    sz = data[0]
    for i in range(1, sz + 1):
        a[i] = data[i]

    for i in range(1, sz):
        # a[i] -> a[i + 1] 인 간선
        adj[a[i]].append(a[i + 1])
        ind[a[i + 1]] += 1

for i in range(1, n + 1):
    if ind[i] == 0:
        q.append(i)

# 위상정렬
while q:
    cur = q.popleft()
    result.append(cur)
    for nxt in adj[cur]:
        ind[nxt] -= 1
        if ind[nxt] == 0:
            q.append(nxt)

flag = 1
for i in range(1, n + 1):
    # indegree가 0이 아닌 값이 있으면
    # 사이클이 존재 -> 위상 정렬 불가
    if ind[i] != 0:
        flag = 0
        break

if flag:
    sys.stdout.write("\n".join(map(str, result)))
else:
    sys.stdout.write("0")
```

# 질문?

# 문제집 / 1766

## 백준 1766 / https://www.acmicpc.net/problem/1766

민오는 1번부터 N번까지 총 N개의 문제로 되어 있는 문제집을 풀려고 한다. 문제는 난이도 순서로 출제되어 있다. 즉 1번 문제가 가장 쉬운 문제이고 N번 문제가 가장 어려운 문제가 된다.

어떤 문제부터 풀까 고민하면서 문제를 훑어보던 민오는, 몇몇 문제들 사이에는 '먼저 푸는 것이 좋은 문제'가 있다는 것을 알게 되었다. 예를 들어 1번 문제를 풀고 나면 4번 문제가 쉽게 풀린다거나 하는 식이다. 민오는 다음의 세 가지 조건에 따라 문제를 풀 순서를 정하기로 하였다.

1. N개의 문제는 모두 풀어야 한다.
2. 먼저 푸는 것이 좋은 문제가 있는 문제는, 먼저 푸는 것이 좋은 문제를 반드시 먼저 풀어야 한다.
3. 가능하면 쉬운 문제부터 풀어야 한다.

예를 들어서 네 개의 문제가 있다고 하자. 4번 문제는 2번 문제보다 먼저 푸는 것이 좋고, 3번 문제는 1번 문제보다 먼저 푸는 것이 좋다고 하자. 만일 4-3-2-1의 순서로 문제를 풀게 되면 조건 1과 조건 2를 만족한다. 하지만 조건 3을 만족하지 않는다. 4보다 3을 충분히 먼저 풀 수 있기 때문이다. 따라서 조건 3을 만족하는 문제를 풀 순서는 3-1-4-2가 된다.

문제의 개수와 먼저 푸는 것이 좋은 문제에 대한 정보가 주어졌을 때, 주어진 조건을 만족하면서 민오가 풀 문제의 순서를 결정해 주는 프로그램을 작성하시오.

### 입력

첫째 줄에 문제의 수 N(1 ≤ N ≤ 32,000)과 먼저 푸는 것이 좋은 문제에 대한 정보의 개수 M(1 ≤ M ≤ 100,000)이 주어진다. 둘째 줄부터 M개의 줄에 걸쳐 두 정수의 순서쌍 A,B가 빈칸을 사이에 두고 주어진다. 이는 A번 문제는 B번 문제보다 먼저 푸는 것이 좋다는 의미이다.

항상 문제를 모두 풀 수 있는 경우만 입력으로 주어진다.

### 출력

첫째 줄에 문제 번호를 나타내는 1 이상 N 이하의 정수들을 민오가 풀어야 하는 순서대로 빈칸을 사이에 두고 출력한다.

# 문제집 / 1766

**아래에 예제에서는**
**4 -> 2**
**3 -> 1 순서로 문제를 풀어야 함**

1. N개의 문제는 모두 풀어야 한다.
2. 먼저 푸는 것이 좋은 문제가 있는 문제는, 먼저 푸는 것이 좋은 문제를 반드시 먼저 풀어야 한다.
3. 가능하면 쉬운 문제부터 풀어야 한다.

### 예제 입력 1 복사

```
4 2
4 2
3 1
```

### 예제 출력 1 복사

```
3 1 4 2
```

# 문제집 / 1766

1. N개의 문제는 모두 풀어야 한다.
2. 먼저 푸는 것이 좋은 문제가 있는 문제는, 먼저 푸는 것이 좋은 문제를 반드시 먼저 풀어야 한다.
3. 가능하면 쉬운 문제부터 풀어야 한다.

## 먼저 푸는 것이 좋은 문제를 먼저 풀어야 하는 것은 위상 정렬로 해결 할 수 있음

### 예제 입력 1 복사

```
4 2
4 2
3 1
```

### 예제 출력 1 복사

```
3 1 4 2
```

# 문제집 / 1766

1. N개의 문제는 모두 풀어야 한다.
2. 먼저 푸는 것이 좋은 문제가 있는 문제는, 먼저 푸는 것이 좋은 문제를 반드시 먼저 풀어야 한다.
3. 가능하면 쉬운 문제부터 풀어야 한다.

## 3번의 조건이 없다면

**4 3 1 2**

**4 2 3 1**

**3 4 2 1**

**3 4 1 2**

### 예제 입력 1 복사

```
4 2
4 2
3 1
```

### 예제 출력 1 복사

```
3 1 4 2
```

# 문제집 / 1766

가능하면 먼저 쉬운 문제를 풀어야 함

처음에 indegree가 0인 값을 큐에 넣으면 3, 4가 들어 감
3, 4에는 우선순위가 없기 때문에 둘 중 아무 값이 먼저 와도 괜찮음

하지만 이 문제에서는 작은 숫자가 먼저 와야 함

예제 입력 1 복사

```
4 2
4 2
3 1
```

예제 출력 1 복사

```
3 1 4 2
```

# 문제집 / 1766

하지만 이 문제에서는 작은 숫자가 먼저 와야 함
-> 큐에 값이 여러 개 있으면 가장 작은 값을 출력해야 함
우선순위 큐

# 문제집 / 1766

C++

```cpp
const ll MAX = 40101;
const ll INF = 1e12;
ll n, m, ind[MAX];
vector <ll> adj[MAX];
priority_queue <ll, vector<ll>, greater<ll>> pq;
```

```cpp
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m;
    while(m--){
        ll s, e; cin >> s >> e;
        adj[s].push_back(e);
        ind[e]++;
    }

    // 우선순위 큐 사용
    for(int i = 1;i <= n;i++){
        if(!ind[i]) pq.push(i);
    }

    while(!pq.empty()){
        ll cur = pq.top(); pq.pop();
        cout << cur << " ";
        for(auto& nxt : adj[cur]){
            if(!--ind[nxt]) pq.push(nxt);
        }
    }

    return 0;
}
```

# 문제집 / 1766

## Python

```python
MAX = 40101
INF = 10**12

n, m = map(int, input().split())
ind = [0] * MAX
adj = [[] for _ in range(MAX)]
pq = []
```

```python
for _ in range(m):
    s, e = map(int, input().split())
    adj[s].append(e)
    ind[e] += 1

# 우선순위 큐 사용
for i in range(1, n + 1):
    if ind[i] == 0:
        heapq.heappush(pq, i)

while pq:
    cur = heapq.heappop(pq)
    print(cur, end=' ')
    for nxt in adj[cur]:
        ind[nxt] -= 1
        if ind[nxt] == 0:
            heapq.heappush(pq, nxt)
```

# 질문?

# 장난감 조립 / 2637

## 백준 2637 / https://www.acmicpc.net/problem/2637

### 문제

우리는 어떤 장난감을 여러 가지 부품으로 조립하여 만들려고 한다. 이 장난감을 만드는데는 기본 부품과 그 기본 부품으로 조립하여 만든 중간 부품이 사용된다. 기본 부품은 다른 부품을 사용하여 조립될 수 없는 부품이다. 중간 부품은 또 다른 중간 부품이나 기본 부품을 이용하여 만들어지는 부품이다.

예를 들어보자. 기본 부품으로서 1, 2, 3, 4가 있다. 중간 부품 5는 2개의 기본 부품 1과 2개의 기본 부품 2로 만들어진다. 그리고 중간 부품 6은 2개의 중간 부품 5, 3개의 기본 부품 3과 4개의 기본 부품 4로 만들어진다. 마지막으로 장난감 완제품 7은 2개의 중간 부품 5, 3개의 중간 부품 6과 5개의 기본 부품 4로 만들어진다. 이런 경우에 장난감 완제품 7을 만드는데 필요한 기본 부품의 개수는 1번 16개, 2번 16개, 3번 9개, 4번 17개이다.

이와 같이 어떤 장난감 완제품과 그에 필요한 부품들 사이의 관계가 주어져 있을 때 하나의 장난감 완제품을 조립하기 위하여 필요한 기본 부품의 종류별 개수를 계산하는 프로그램을 작성하시오.

### 입력

첫째 줄에는 자연수 N(3 ≤ N ≤ 100)이 주어지는데, 1부터 N-1까지는 기본 부품이나 중간 부품의 번호를 나타내고, N은 완제품의 번호를 나타낸다. 그리고 그 다음 줄에는 자연수 M(3 ≤ M ≤ 100)이 주어지고, 그 다음 M개의 줄에는 어떤 부품을 완성하는데 필요한 부품들 간의 관계가 3개의 자연수 X, Y, K로 주어진다. 이 뜻은 "중간 부품이나 완제품 X를 만드는데 중간 부품 혹은 기본 부품 Y가 K개 필요하다"는 뜻이다. 두 중간 부품이 서로를 필요로 하는 경우가 없다.

### 출력

하나의 완제품을 조립하는데 필요한 기본 부품의 수를 한 줄에 하나씩 출력하되(중간 부품은 출력하지 않음), 반드시 기본 부품의 번호가 작은 것부터 큰 순서가 되도록 한다. 각 줄에는 기본 부품의 번호와 소요 개수를 출력한다.

정답은 2,147,483,647 이하이다.

# 장난감 조립 / 2637

**부품의 개수 N과 부품들 사이의 관계의 수 M이 주어짐**
**각 관계는 E S C가 주어지고 부품 E를 만들기 위해서는**
**부품 S가 C개 필요함**

예제 입력 1 복사

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

예제 출력 1 복사

```
1 16
2 16
3 9
4 17
```

# 장난감 조립 / 2637

어떤 부품을 만들기 위해 다른 부품이 필요한 것이 없는 것들을
기본 부품이라고 함
부품 N을 만들기 위해서 기본 부품이 몇 개 필요한지 구하는 문제

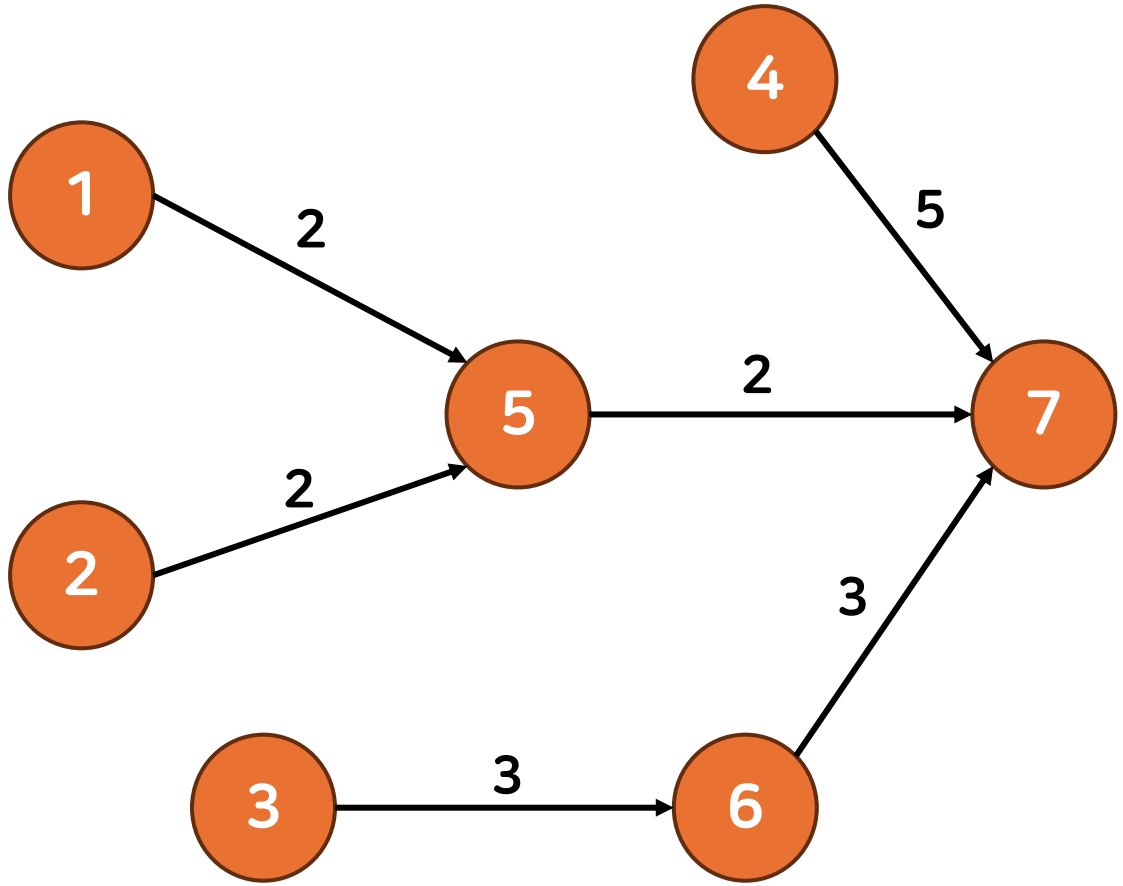
예제 입력 1 복사

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

예제 출력 1 복사

```
1 16
2 16
3 9
4 17
```

# 장난감 조립 / 2637

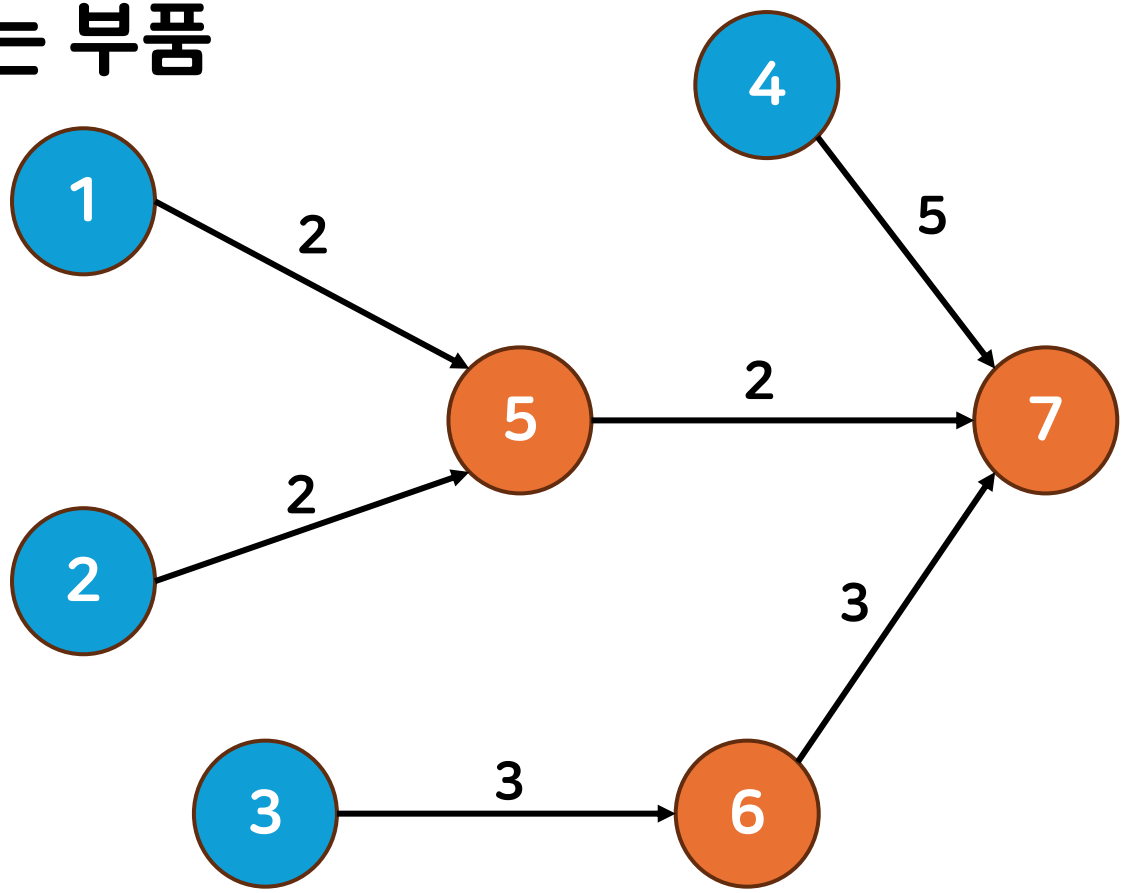


# 장난감 조립 / 2637

기본 부품

자신을 구성하는 다른 부품이 없는 부품

= indegree가 0




예제 출력 1 복사

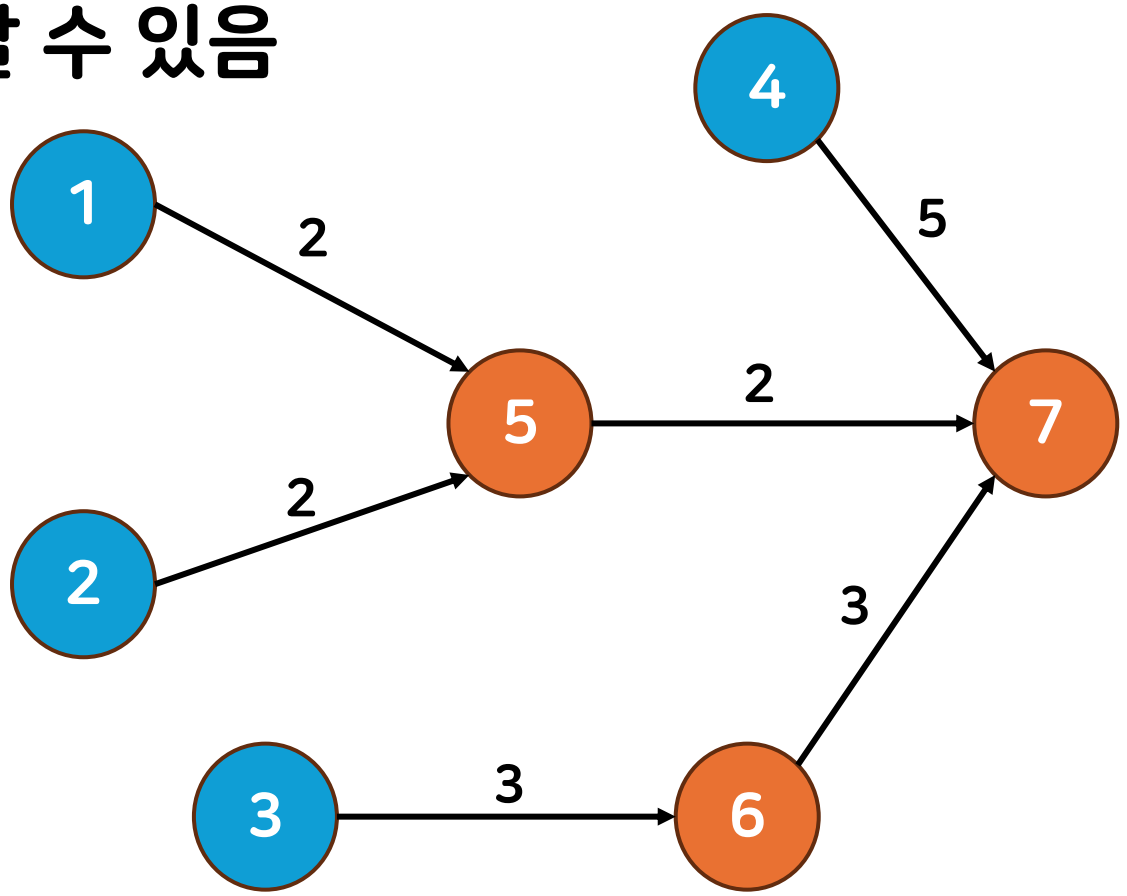
```
1 16
2 16
3 9
4 17
```

# 장난감 조립 / 2637

각 부품을 만드는데 기본 부품의 개수를 알면
다음 부품을 만드는데 필요한 개수를 알 수 있음

5를 만드는데 필요한 기본 부품

1을 만드는데 필요한 기본 부품 X 2
2를 만드는데 필요한 기본 부품 X 2



# 장난감 조립 / 2637

DP[i][j] = i번째 부품을 만드는데 필요한 j번째 부품의 개수
(단 j는 기본 부품)

각 기본 부품들은 자기 자신 한 개로 만들 수 있음

Base Case
DP[i][i] = 1 (단 i는 기본 부품)

# 장난감 조립 / 2637

DP[i][j] = i번째 부품을 만드는데 필요한 j번째 부품의 개수
(단 j는 기본 부품)

DP 테이블을 채우기 위해서는 순서를 알아야 함
-> 위상 정렬 순서로 따라가면 됨

# 장난감 조립 / 2637

DP[i][j] = i번째 부품을 만드는데 필요한 j번째 부품의 개수
(단 j는 기본 부품)

다음 부품을 만드는데 필요한 기본 부품의 개수는
현재 부품을 만드는데 필요한 기본 부품의 개수에 비용을 곱해주면 됨

DP[nxt][j] += DP[cur][j] * cost (단 j는 기본 부품)

# 장난감 조립 / 2637

C++

```
ll n, m, ind[MAX], dp[MAX][MAX];
vector <ll> basic;
queue <ll> q;
vector <pll> adj[MAX];
```

```
int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n >> m;
    while(m--){
        ll e, s, c; cin >> e >> s >> c;
        adj[s].push_back({e, c});
        ind[e]++;
    }

    for(int i = 1;i <= n;i++){
        if(ind[i]) continue;
        // ind[i] == 0 이면
        // 큐에 넣고 기본 부품으로 지정
        q.push(i);
        basic.push_back(i);

        // 기본 부품을 만드는데는 자기 자신 1개가 필요함
        dp[i][i] = 1;
    }

    while(!q.empty()){
        auto cur = q.front(); q.pop();

        for(auto& [nxt, co] : adj[cur]){
            // dp[nxt]를 만드는데 드는 부품은 dp[cur] * co 만큼 추가로 들음
            // 각 부품 1 ~ N 까지 반복해야 함
            for(int i = 1;i <= n;i++) dp[nxt][i] += co * dp[cur][i];
            if(!--ind[nxt]) q.push(nxt);
        }
    }

    for(auto& i : basic){
        // 기본 부품에 대해서만 개수 출력
        cout << i << " " << dp[n][i] << "\n";
    }

    return 0;
}
```

# 장난감 조립 / 2637

Python

```python
import sys
from collections import deque
input = sys.stdin.readline

n = int(input())
m = int(input())

ind = [0] * (n + 1)
dp = [[0] * (n + 1) for _ in range(n + 1)]
basic = []
q = deque()
adj = [[] for _ in range(n + 1)]
```

```python
for _ in range(m):
    e, s, c = map(int, input().split())
    adj[s].append((e, c))
    ind[e] += 1

for i in range(1, n + 1):
    if ind[i]:
        continue
    # ind[i] == 0 이면
    # 큐에 넣고 기본 부품으로 지정
    q.append(i)
    basic.append(i)
    # 기본 부품을 만드는데는 자기 자신 1개가 필요함
    dp[i][i] = 1


while q:
    cur = q.popleft()
    for nxt, co in adj[cur]:
        # dp[nxt]를 만드는데 드는 부품은 dp[cur] * co 만큼 추가로 들음
        # 각 부품 1 ~ N 까지 반복해야 함
        for i in range(1, n + 1):
            dp[nxt][i] += dp[cur][i] * co
        ind[nxt] -= 1
        if ind[nxt] == 0:
            q.append(nxt)

# 기본 부품에 대해서만 개수 출력
for i in basic:
    print(i, dp[n][i])
```

# 질문?

# 추가로 배울만한 것들

필수

분할 정복 / https://blog.naver.com/kks227/220776241154

백트래킹 / https://blog.naver.com/kks227/220786417910

누적 합 / https://blog.naver.com/kks227/220787178657

투 포인터 / https://blog.naver.com/kks227/220795165570

# 추가로 배울만한 것들

심화

플로이드 / https://m.blog.naver.com/kks227/220797649276

Union find / https://blog.naver.com/kks227/220791837179

MST / https://m.blog.naver.com/kks227/220799105543

Trie / https://blog.naver.com/kks227/220938122295

# 꾸준히 하는 법

목표를 잡기

안하고 싶을 때가 굉장히 많음

해야 하는 이유를 만들자

어떤 목표를 잡을까?

내가 했던 방식 5개

# 꾸준히 하는 법

## 1. 단계별로 풀어보기
### 코테를 위해서는 36까지 풀어보면 됨

| 단계 | 제목 | 설명 | 정보 | 총 문제 | 내가 맞은 문제 |
|---|---|---|---|---|---|
| 1 | 입출력과 사칙연산 | 입력, 출력과 사칙연산을 연습해 봅시다. Hello World! | 완료 | 13 | 13 |
| 2 | 조건문 | if 등의 조건문을 사용해 봅시다. | 완료 | 7 | 7 |
| 3 | 반복문 | for, while 등의 반복문을 사용해 봅시다. | 완료 | 12 | 12 |
| 4 | 1차원 배열 | 배열을 사용해 봅시다. | 완료 | 10 | 10 |
| 5 | 문자열 | 문자열을 다루는 문제들을 해결해 봅시다. | 완료 | 11 | 11 |
| 6 | 심화 1 | 지금까지의 프로그래밍 문법으로 더 어려운 문제들을 풀어봅시다. | 완료 | 8 | 8 |
| 7 | 2차원 배열 | 배열 안에 배열이 있다면 어떨까요? 2차원 배열을 만들어 봅시다. | 완료 | 4 | 4 |
| 8 | 일반 수학 1 | 수학적 사고력을 길러 봅시다. | 도전 중 | 7 | 5 |
| 9 | 약수, 배수와 소수 | 약수와 배수는 정수론의 시작점이라고 할 수 있습니다. | 완료 | 6 | 6 |
| 10 | 기하: 직사각형과 삼각형 | 간단한 도형으로 기하 문제풀이를 시작해 봅시다. | 도전 중 | 8 | 2 |
| 11 | 시간 복잡도 | 프로그램의 정확한 실행 시간을 예측하기는 매우 어렵습니다. 하지만 시간 복잡도를 사용하여 대략적인 예측은 가능합니다. | | 7 | 0 |
| 12 | 브루트 포스 | 가장 간단한 알고리즘인, 모든 경우의 수를 검사하는 브루트 포스 알고리즘을 배워 봅시다. | 도전 중 | 6 | 5 |
| 13 | 정렬 | 배열의 원소를 순서대로 나열하는 알고리즘을 배워 봅시다. | 완료 | 11 | 11 |
| 14 | 집합과 맵 | 특정 원소가 속해 있는지 빠르게 찾거나, 각 원소에 대응되는 원소를 빠르게 찾는 자료구조를 배워 봅시다. | 도전 중 | 8 | 6 |
| 15 | 약수, 배수와 소수 2 | 정수론의 세계로 조금 더 들어가 봅시다. | 도전 중 | 9 | 6 |
| 16 | 스택, 큐, 덱 | 스택, 큐, 덱 자료구조를 사용하여 문제를 해결해 봅시다. | 도전 중 | 11 | 6 |
| 17 | 조합론 | 경우의 수를 세어 봅시다. | 도전 중 | 5 | 3 |
| 18 | 심화 2 | 👑 | 도전 중 | 5 | 2 |
| 19 | 재귀 | 재귀함수를 다뤄 봅시다. | 도전 중 | 7 | 6 |
| 20 | 백트래킹 | 모든 경우를 탐색하는 백트래킹 알고리즘을 배워 봅시다. | 완료 | 8 | 8 |
| 21 | 동적 계획법 1 | 기초적인 동적 계획법 문제들을 풀어봅시다. | 완료 | 16 | 16 |
| 22 | 누적 합 | 부분구간 안에 있는 수들의 합을 빠르게 구해 봅시다. | 도전 중 | 6 | 5 |
| 23 | 그리디 알고리즘 | 특정 상황에서 성립하는 그리디 알고리즘을 배워 봅시다. | 완료 | 5 | 5 |
| 24 | 분할 정복 | 재귀를 응용하는 알고리즘, 분할 정복을 익혀 봅시다. | 도전 중 | 9 | 8 |
| 25 | 이분 탐색 | 이분 탐색 알고리즘을 배워 봅시다. | 완료 | 7 | 7 |
| 26 | 우선순위 큐 | 가장 작은/큰 원소를 뽑는 자료구조를 배워 봅시다. | 도전 중 | 6 | 5 |
| 27 | 동적 계획법 2 | 조금 더 어려운 동적 계획법 문제를 풀어 봅시다. | 완료 | 7 | 7 |
| 28 | 스택 2 | 스택을 사용하여 더욱 어려운 문제를 해결해 봅시다. | 완료 | 5 | 5 |
| 29 | 그래프와 순회 | 그래프를 배우고, 그래프를 순회하는 알고리즘을 배워 봅시다. | 도전 중 | 16 | 14 |
| 30 | 위상 정렬 | 간선에 방향이 있는 그래프의 정점을 나열해 역방향이 없게 만드는 알고리즘을 다뤄 봅시다. | 완료 | 3 | 3 |
| 31 | 최단 경로 | 그래프의 간선에 가중치가 없으면 BFS로 최단거리를 찾을 수 있습니다. 가중치가 있다면 어떨까요? | 완료 | 7 | 7 |
| 32 | 투 포인터 | 투 포인터 알고리즘과 meet in the middle 알고리즘을 배워 봅시다. | 완료 | 5 | 5 |
| 33 | 동적 계획법과 최단거리 역추적 | 지금까지는 최솟값, 최댓값, 최단거리만 찾았습니다. 이번에는 실제 최적해와 최단경로를 찾아 봅시다. | 완료 | 9 | 9 |
| 34 | 트리 | 대표적인 그래프 종류 중 하나인 트리를 다뤄 봅시다. | 완료 | 7 | 7 |
| 35 | 유니온 파인드 1 | 유니온 파인드(또는 disjoint set, 상호 배타적 집합, ...) 자료구조를 배워 봅시다. | 완료 | 4 | 4 |
| 36 | 최소 신장 트리 | 최소 비용으로 그래프의 모든 정점을 연결해 봅시다. | 도전 중 | 6 | 5 |
| 37 | 해 구성하기 | 주어진 조건을 만족하는 답안을 실제로 만들어 봅시다. | 도전 중 | 10 | 7 |
| 38 | 트리에서의 동적 계획법 | 트리에 동적 계획법을 적용해 봅시다. | 도전 중 | 5 | 4 |
| 39 | 기하 2 | 조금 더 어려운 기하 문제를 풀어 봅시다. | 도전 중 | 10 | 9 |
| 40 | 동적 계획법 3 | 비트마스크를 배우고 동적 계획법에 적용해 봅시다. 그 후에는 선형이 아니라 원형으로 구성된 문제, 그리고 DAG 에서의 문제를 다룹니다. | 도전 중 | 8 | 7 |
| 41 | 인터랙티브와 투 스텝 1 | 채점기와 상호 작용하면서 문제를 풀어봅시다. | 도전 중 | 8 | 2 |
| 42 | 유니온 파인드 2 | 유니온 파인드를 이용해 다양한 문제를 해결해 봅시다. | 도전 중 | 7 | 6 |
| 43 | 문자열 알고리즘 1 | KMP 알고리즘과 트라이 자료구조, 문자열 해싱을 다뤄 봅시다. | 도전 중 | 8 | 5 |
| 44 | 최소 공통 조상 | 트리에서 두 정점의 최소 공통 조상을 구하는 자료구조를 배워 봅시다. | 완료 | 5 | 5 |
| 45 | 강한 연결 요소 | Strongly connected component를 다뤄 봅시다. | 완료 | 8 | 8 |
| 46 | 세그먼트 트리 | 구간 쿼리를 효율적으로 수행하는 자료구조를 배워 봅시다. | 완료 | 8 | 8 |
| 47 | 스위핑 | 스위핑 알고리즘을 배워 봅시다. | 완료 | 6 | 6 |
| 48 | 동적 계획법 4 | 동적 계획법의 세계는 끝이 없습니다. | 도전 중 | 13 | 8 |
| 49 | 기하 3 | 모든 점을 포함하는 가장 작은 볼록 다각형을 만들어 봅시다. | 도전 중 | 6 | 5 |
| 50 | 어려운 구간 쿼리 | 세그먼트 트리 with lazy propagation과 Mo's algorithm을 배워 봅시다. | 도전 중 | 13 | 9 |
| 51 | 이분 매칭 | 이분 매칭 알고리즘에 대해 배워 봅시다. | 도전 중 | 6 | 4 |
| 52 | 네트워크 플로우 1 | 네트워크 플로우 알고리즘에 대해 알아봅시다. | 도전 중 | 10 | 3 |
| 53 | 네트워크 플로우 2 | 최소 비용으로 최대 유량을 흘려 봅시다. | 도전 중 | 4 | 2 |

# 꾸준히 하는 법

## 2. solved.ac class

코테를 위해서는 Class 4 까지 다 밀고 Class 5 찍먹



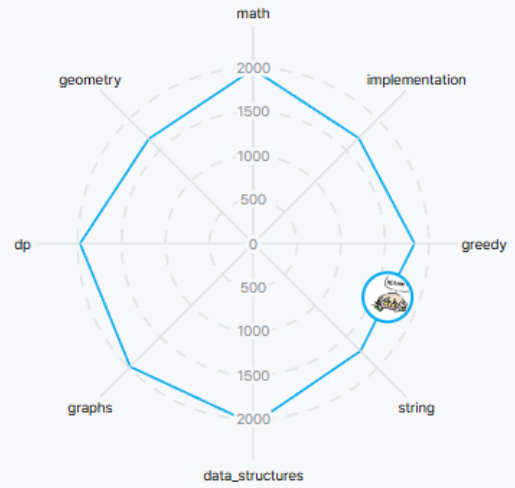
총 48문제, 에센셜 24문제

| | 미해결 | 해결 | 취득 조건 | 진행도 |
|---|---|---|---|---|
| CLASS 8 | 3 | 17 | 목록 중 20문제 이상 | |
| CLASS 8* | 14 | 10 | 에센셜 24문제 | |
| CLASS 8** | 31 | 17 | 48문제 | |

# 꾸준히 하는 법

## 3. solved.ac 팔각형 만들기
### 자신이 취약한 분야를 알 수 있음
### 한때 기하, 문자열 문제만 풀음

태그 분포




| 태그 | 문제 | | 레이팅 |
|---|---|---|---|
| #자료 구조 | 302 | 15.5% | 2 2054 |
| #그래프 이론 | 378 | 19.4% | 3 1982 |
| #수학 | 471 | 24.1% | 3 1982 |
| #다이나믹 프로그래밍 | 387 | 19.8% | 3 1972 |
| #세그먼트 트리 | 69 | 3.5% | 3 1904 |
| #그리디 알고리즘 | 225 | 11.5% | 4 1837 |
| #정렬 | 194 | 9.9% | 4 1784 |
| #정수론 | 134 | 6.9% | 4 1782 |
| #트리 | 83 | 4.3% | 4 1781 |
| #그래프 탐색 | 220 | 11.3% | 4 1780 |

# 꾸준히 하는 법

## 4. 좋은 대회 하나 잡고 풀어 보기

한때 KOI(한국 정올) 골드 문제만 풀음

현재는 USACO(미국 정올) 골1 ~ 골3 문제를 푸는 중

| 출처 | 하위 출처 | 맞은 문제 | 채점 가능한 문제 | 총 문제 |
|---|---|---|---|---|
| 한국정보올림피아드 | 36 | 297 | 514 | 522 |
| USA Computing Olympiad | 28 | 192 | 1197 | 1244 |

# 꾸준히 하는 법

## 4. 좋은 대회 하나 잡고 풀어 보기
## https://koosaga.com/217

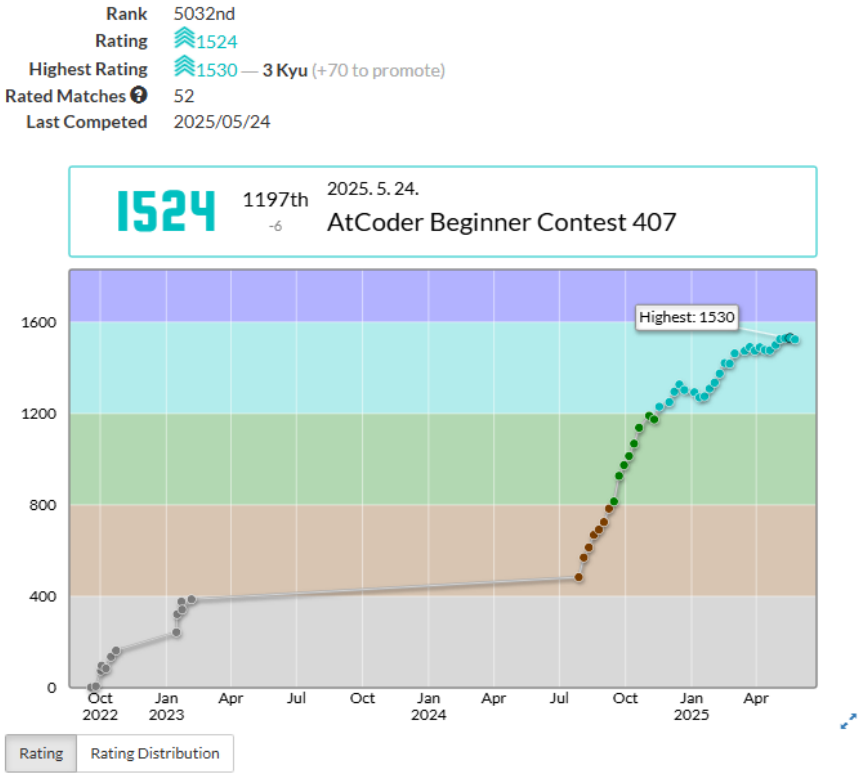
- **추천** USA Computing Olympiad: 전통의 소 농장은 여전히 훌륭한 트레이닝 사이트고 특별히 거를 게 없습니다. 특히 튜토리얼 삼을 문제가 많아 PS 입문/초심자 분들에게 강력히! 추천합니다. 다만 최상위권한테는 약간 애매한듯. 모든 문제에 대해서 풀이를 제공해 주고, 채점도 BOJ 및 공식 사이트가 잘 해줍니다. 겨울에는 온라인 대회를 열어주기 때문에 Online Contest와도 겹치네요.

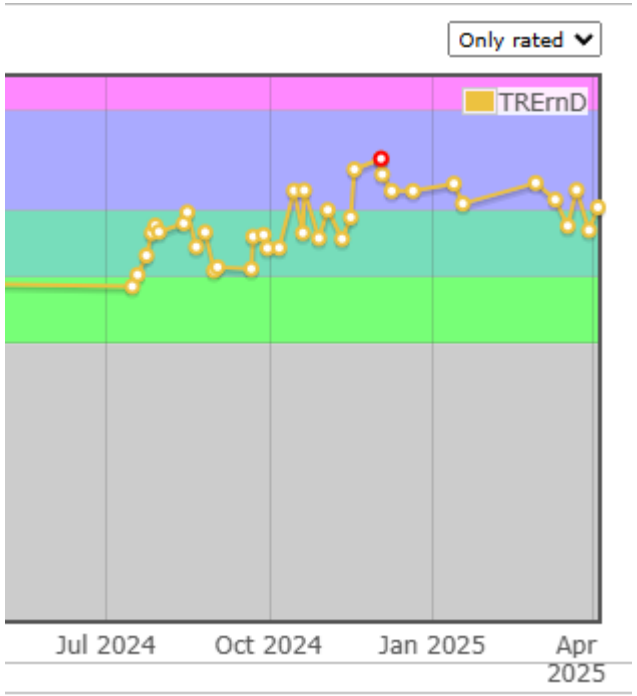
- **추천** Croatian Open Contest in Informatics : 역시 전통의 COCI는 풀이도 항상 제공해 주고 문제도 괜찮습니다. 요즘은 풀이 제공도 불성실하고 문제 수준도 애매한 것 같으나.. 아무튼 얘는 USACO랑 컨셉이 비슷합니다. BOJ에 여전히 문제가 많고, 특히 그 중 대다수가 번역이 되어 있어서, 업솔빙은 여전히 추천. 역시 겨울에는 온라인 대회를 열어주기 때문에 Online Contest와도 겹치네요.

- **추천** 한국정보올림피아드 불친절해서 그렇지 문제는 꽤 괜찮습니다. 국산 사랑해 줍시다.

# 꾸준히 하는 법

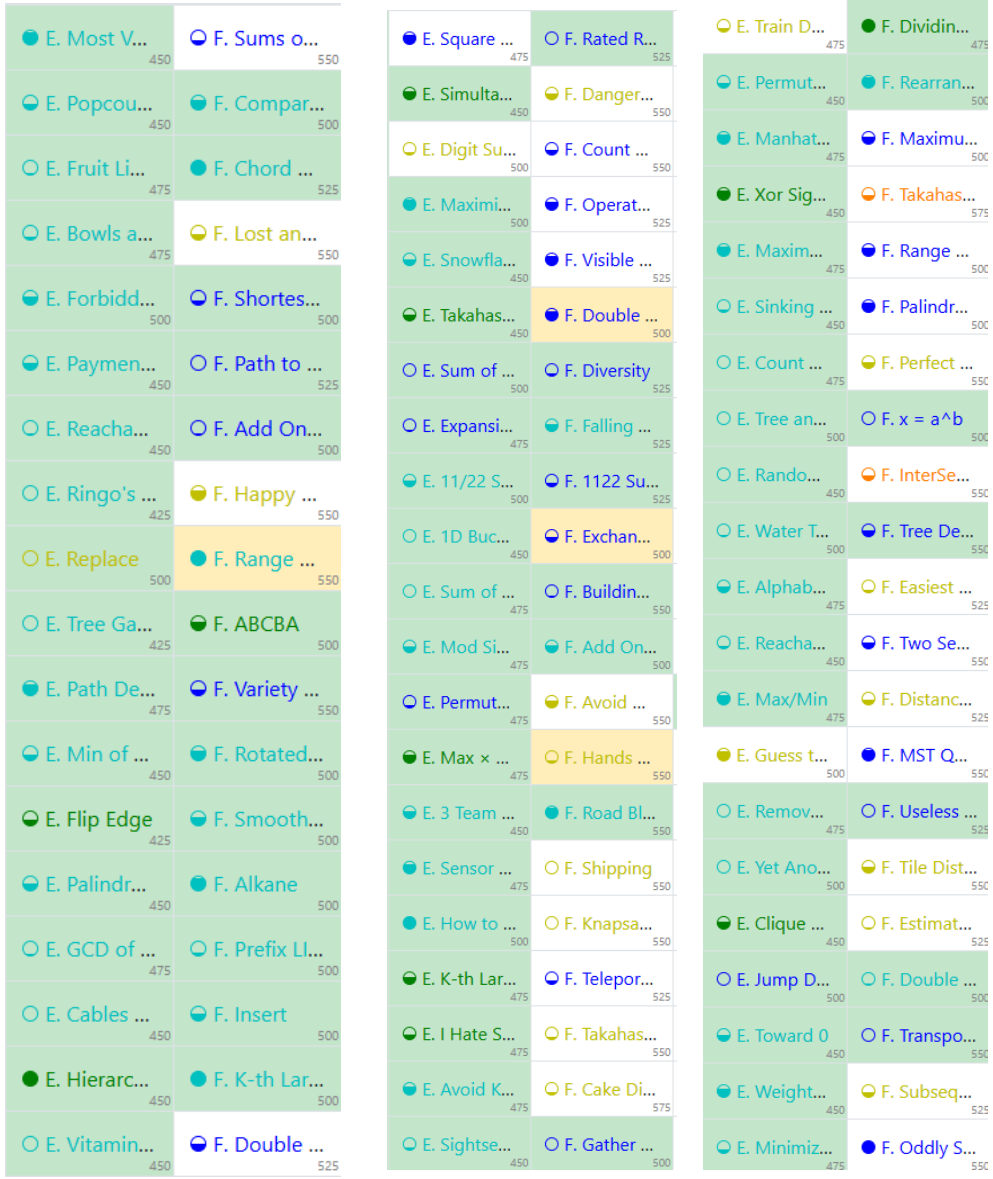
## 5. CP 티어를 목표로 공부하기
### Atcoder, Codeforces 같은 온라인 대회 참여



# 꾸준히 하는 법

## 5. CP 티어를 목표로 공부하기
Atcoder 민트를 달기 위해서
그 티어에 맞는 문제를 찾아서 풀음



# 고생하셨습니다