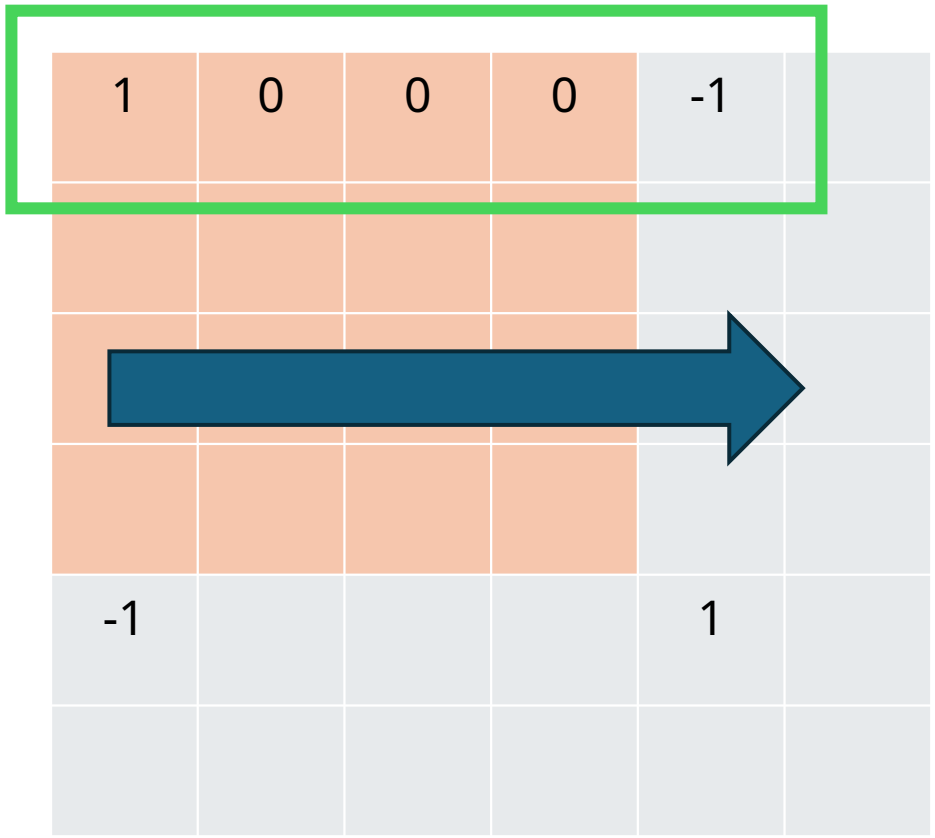
week 8

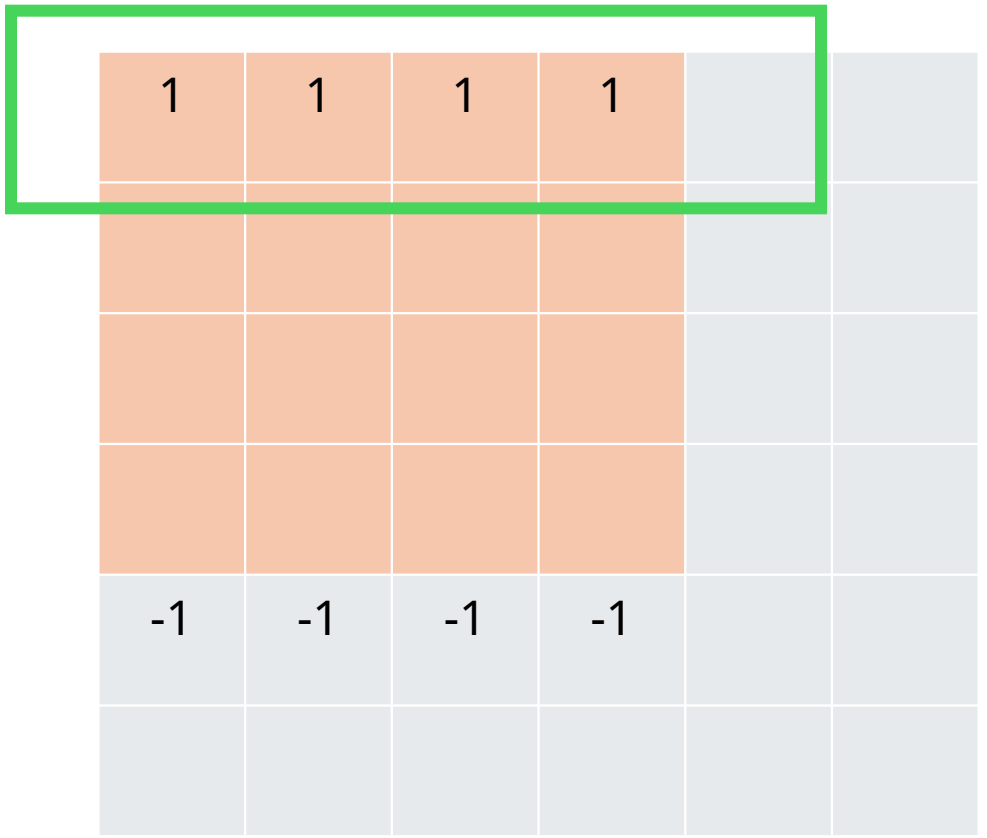
2024-08-22

imos 법（いもす法）추가

초록 네모 친 부분이 1 차원 imos 임

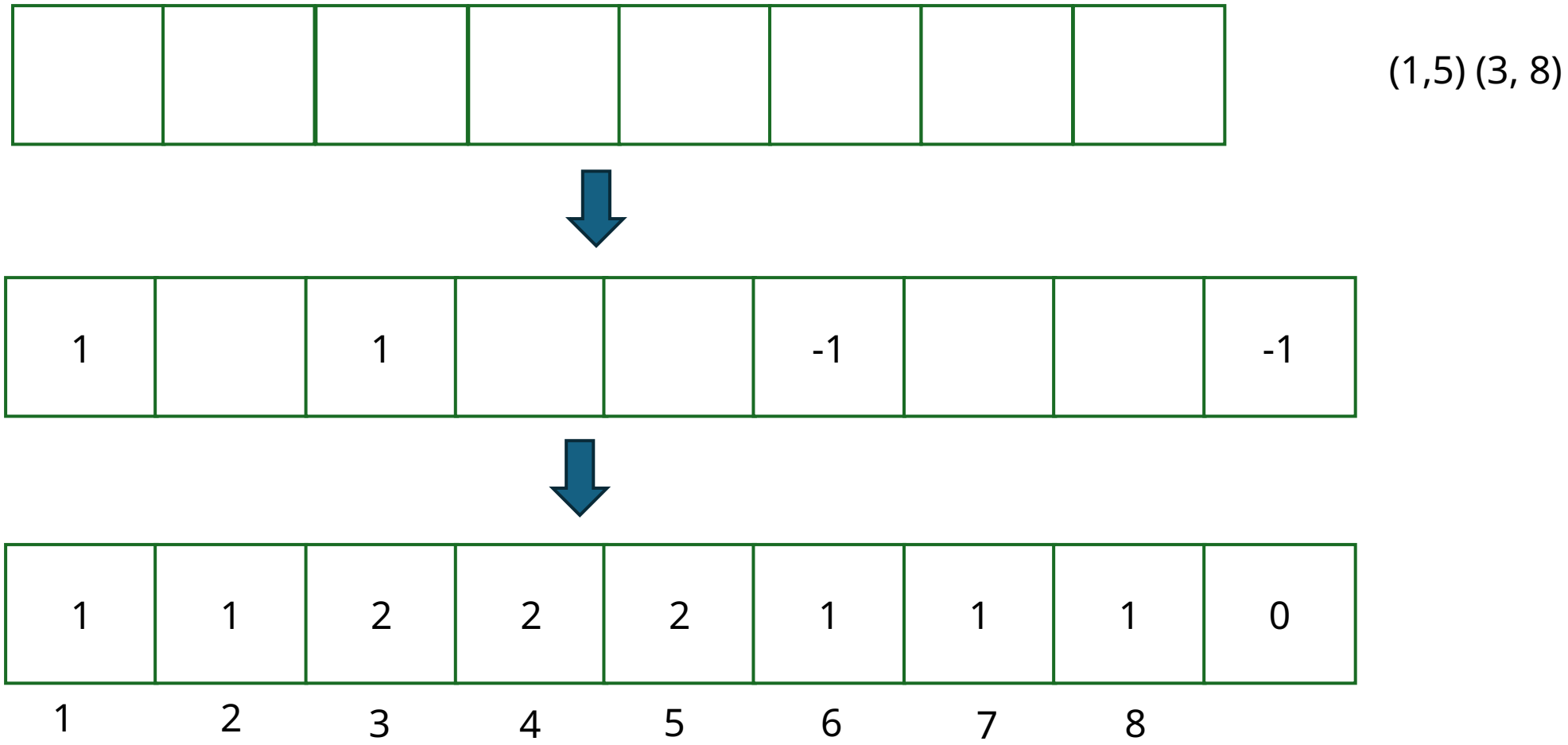
시작점에 +1,
끝점 뒷칸에 -1





imos 법 (いもす法) 추가

1 차원 imos



imos 법 (いもす法) 추가

원래는 저렇게 채워주는게 맞음
그런데 저렇게 전부 다 채워버리면 시간이 오래 걸리는 경우가 있음

만약 강의는 두갠데 시간이 (1, 10^10), (10^5, 10^12) 요런식 .
강의가 두개라면 한 번 겹친다는 것이 쉽게 확인 가능하지만 많다면
imos 를 사용하여 전부 더하고 누적합을 구하기에 시간이 너무 많이걸림

이때 값이 변경되는 지점만 계산해 줄 수 있음
(범위가 크고 개수가 비교적 작은 경우 -> 좌표 압축)

imos 법 (いもす法) 추가

11000: 강의실 배정

Si 에 시작해서 Ti 에 끝나는 N 개의 수업이 주어진다 .

최소의 강의실을 사용해서 모든 수업을 가능하게 해야 함 .

imos 법 (いもす法) 추가

지난번에 사용한 풀이는 일종의 최적화를 거친 것
(필요하지 않은 범위의 값들은 신경쓰지 않는다)

```
for(int i = 0; i < n; i++) {
    int s, e;
    cin >> s >> e;
    a[s] += 1;
    a[e] -= 1;
}
```

```
for(int i = 0; i < n; i++) {
    int s, e;
    cin >> s >> e;
    a.emplace_back(s, 1);
    a.emplace_back(e, -1);
}
sort(a.begin(), a.end());
```

최적화 전 후

```
int ans = 0;
for(int i = 1; i <  범위 ; i++) {
    p[i] = p[i – 1] + a[i];
    ans = max(ans, p[i])
}
```

```
int ans = 0;
int cnt = 0;
for(auto [t1, t2] : a) {
    cnt += t2;
    ans = max(ans, cnt);
}
```

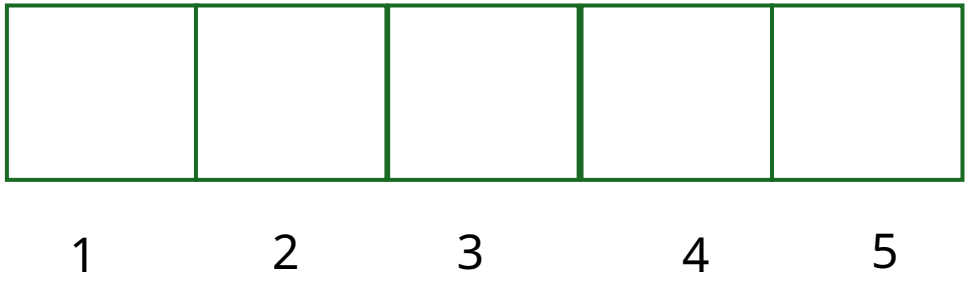
알고있는 경우와 다르게
끝점 +1 인덱스에 -1 을 하지 않고
끝점 인덱스에 -1 을 했는데 ??

imos 법 (いもす法) 추가

알고있는 것과 다르게
끝점 +1 인덱스에 -1 을 하지 않고 바로 끝점 인덱스에 -1 을 한 것은
선분과 배열의 차이 때문임

(1, 3) (3, 5) 를 선분으로 나타내면

배열로 나타내면

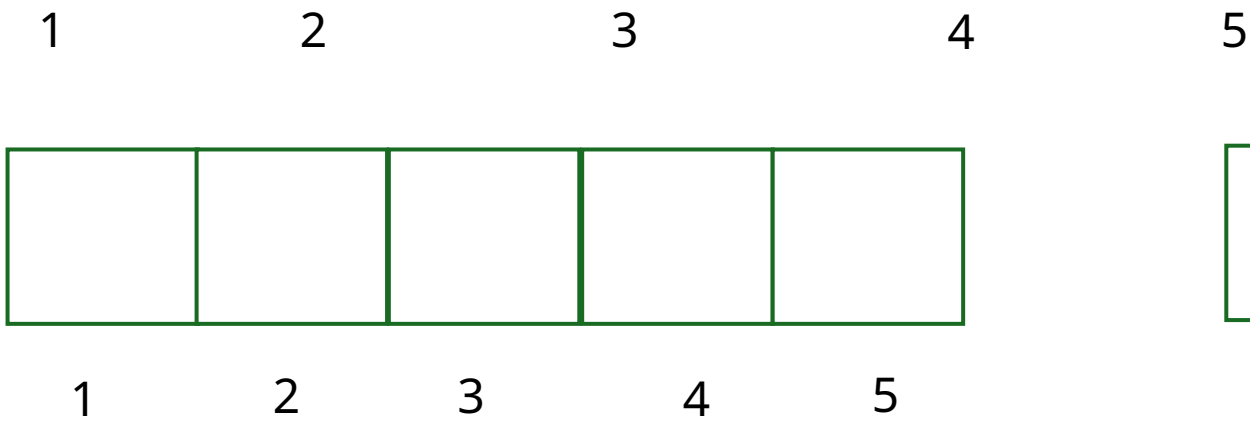


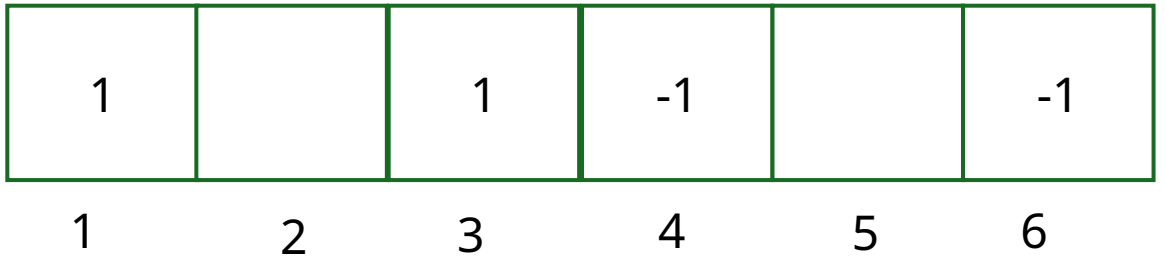
이다 .

문제에서 강의가 끝남과 동시에
시작될 수 있다고 하였으므로
강의는 곧 선분과 같다고 생각할 수 있음
(1,3) (3, 5) 가 안 겹침 .

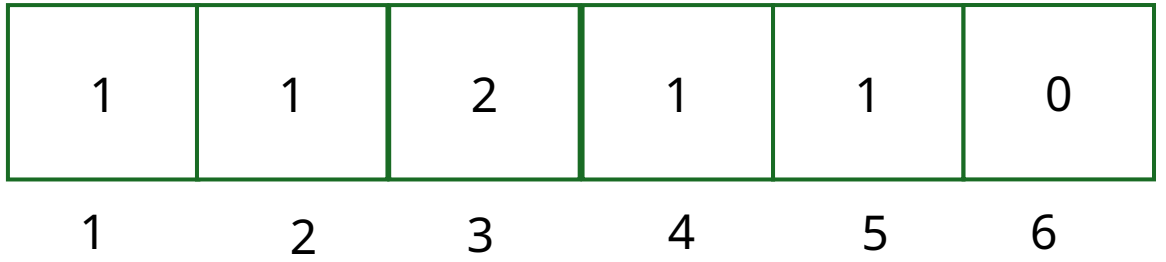
imos 법 (いもす法) 추가

그렇게 imos 를 적용해서 계산해 보면 ...

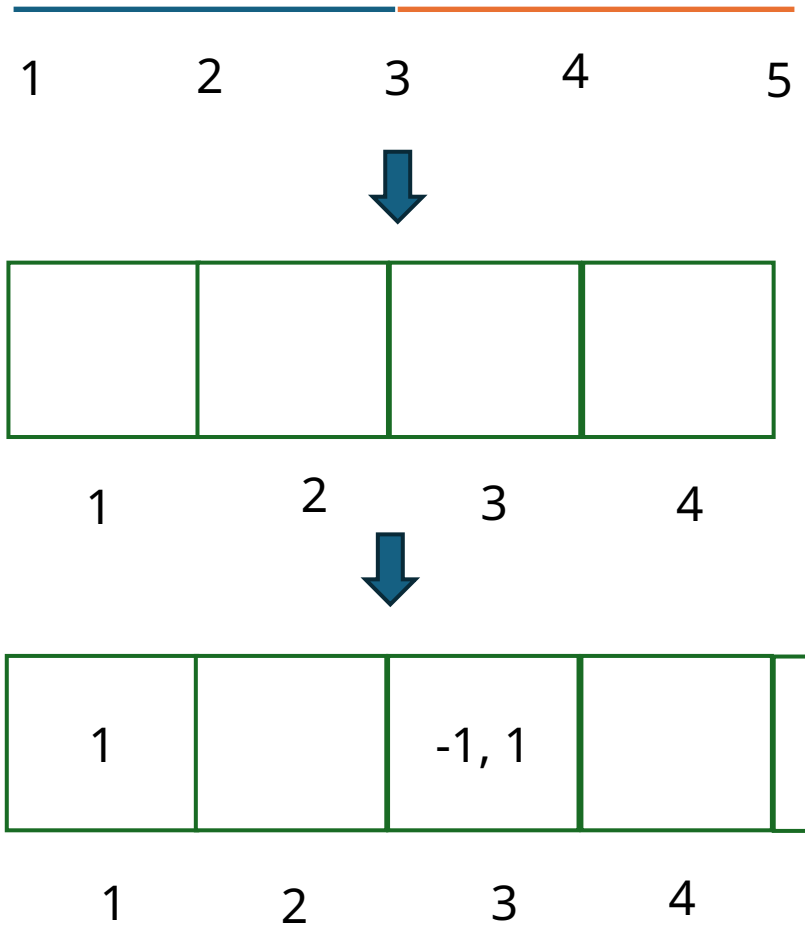




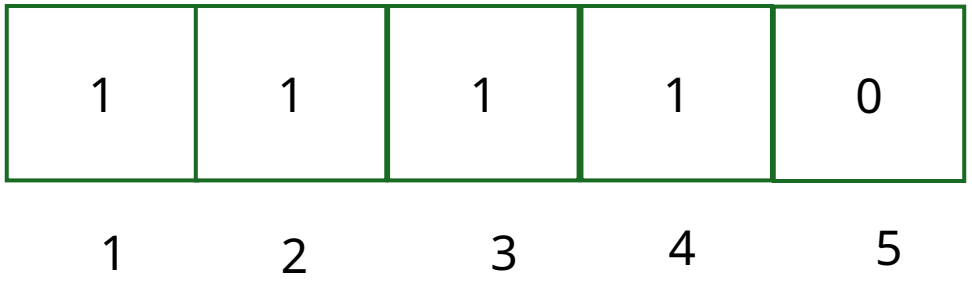
원하는 결과값이 아니다
겹치는 부분이 없는데 2 가 나옴



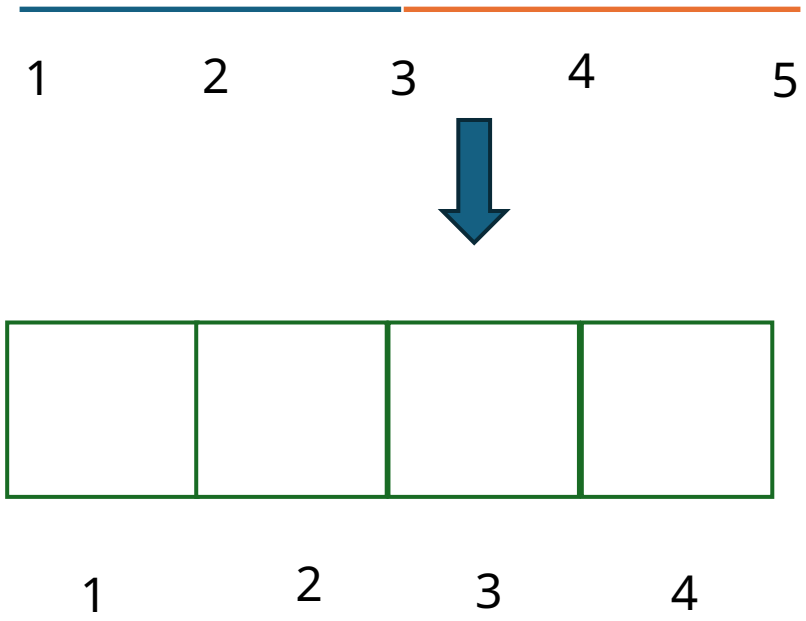
imos 법 (いもす法) 추가



우리는 이런식으로 볼 거임
주어진 선분좌표를 바로 배열에 매칭하지 않음



imos 법 (いもす法) 추가



이를 적용하기 위해선 끝점을
앞으로 한칸씩 당겨주면 됨 .
(1, 3) (3, 5) => (1, 2) (3, 4)

imos 는 끝점 한칸 뒤에 -1 을 해야 하므로
(1, 2) -> 인덱스 1 에 +1, 인덱스 3 에 -1,

따라서 처음 입력의 끝점에 -1 하면됨

imos 법 (いもす法) 추가

```
for(int i = 0; i < n; i++) {
    int s, e;
    cin >> s >> e;
    a[s] += 1;
    a[e] -= 1;
}
```

```
for(int i = 0; i < n; i++) {
    int s, e;
    cin >> s >> e;
    a.emplace_back(s, 1);
    a.emplace_back(e, -1);
}
sort(a.begin(), a.end());
```

그렇게 하면 왼쪽 코드가 나옴

```
int ans = 0;
for(int i = 0; i <  범위 ; i++) {
    if(i == 0) p[i] = a[i];

    p[i] = p[i – 1] + a[i];
    ans = max(ans, p[i])
}
```

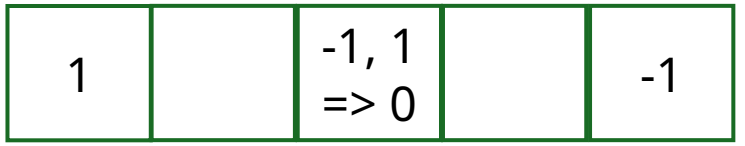
```
int ans = 0;
int cnt = 0;
for(auto [t1, t2] : a) {
    cnt += t2;
    ans = max(ans, cnt);
}
```

imos 법 (いもす法) 추가

```
for(int i = 0; i < n; i++) {
    int s, e;
    cin >> s >> e;
    a[s] += 1;
    a[e] -= 1;
}
```

```
for(int i = 0; i < n; i++) {
    int s, e;
    cin >> s >> e;
    a.emplace_back(s, 1);
    a.emplace_back(e, -1);
}
sort(a.begin(), a.end());
```

오른쪽 위 코드에서
무작위로 입력받은
시간 범위를 정렬해서



이런식으로 나열한다고 생각할 수 이씀

```
int ans = 0;
for(int i = 0; i <  범위 ; i++) {
    if(i == 0) p[i] = a[i];

    p[i] = p[i – 1] + a[i];
    ans = max(ans, p[i])
}
```

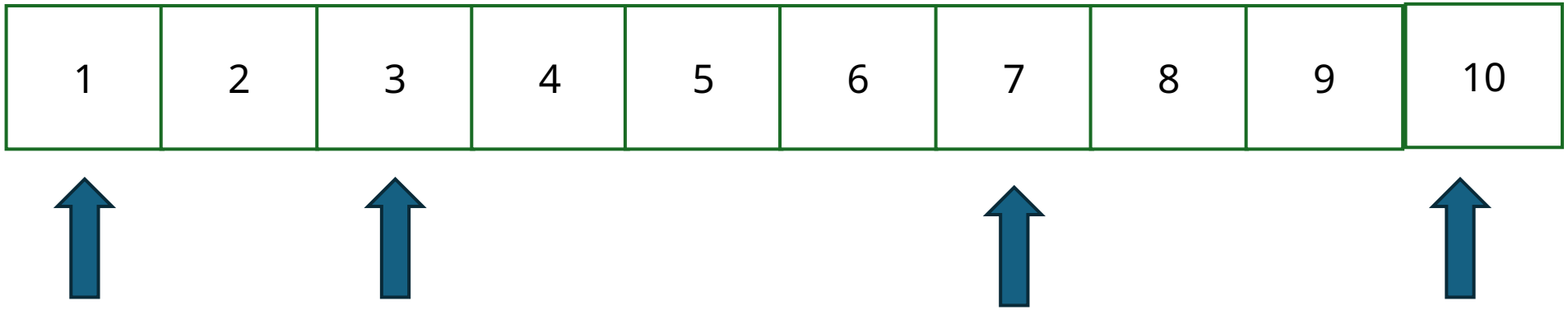
```
int ans = 0;
int cnt = 0;
for(auto [t1, t2] : a) {
    cnt += t2;
    ans = max(ans, cnt);
}
```

imos 법 (いもす法) 추가

예를 들어 아래와 같이 배열되었다고 할 때

(3, 7)

(1, 10)



오른쪽 아래 코드에서
for 문을 돌며 이 부분들을 확인하는 거임

cnt 에 현재 값을 담아서 ans 를 갱신한다 .

2213: 트리의 독립집합

그래프 , 트리 , 트리 DP

인접하지 않은 정점들을 골라서 최대값 만들기

2213: 트리의 독립집합

https://csacademy.com/app/graph_editor/

간선 입력하면

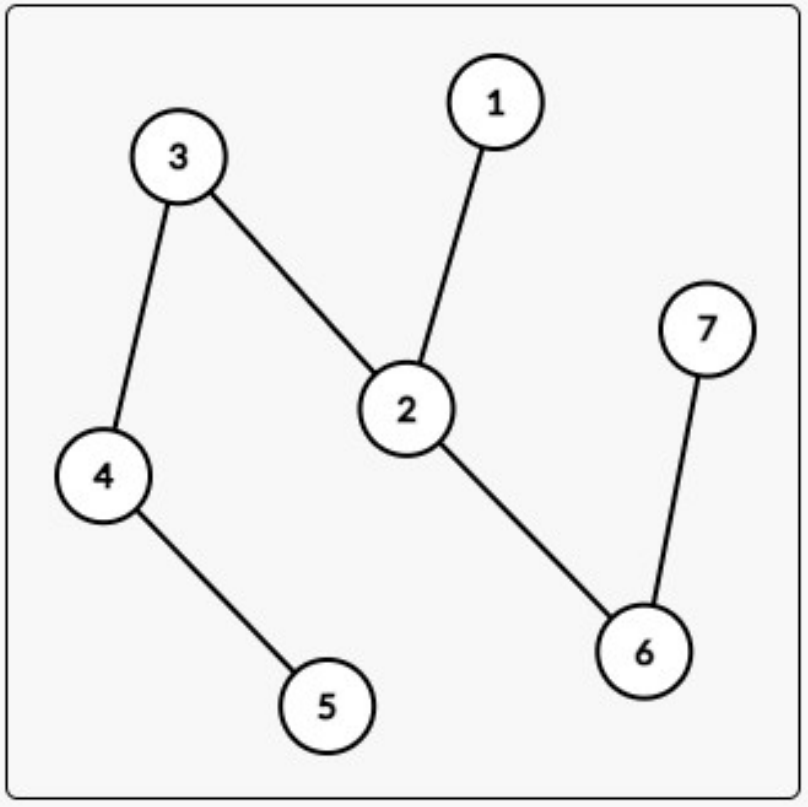
그래프
대신 그려주는
사이트

Node Count:

1	7

Graph Data:

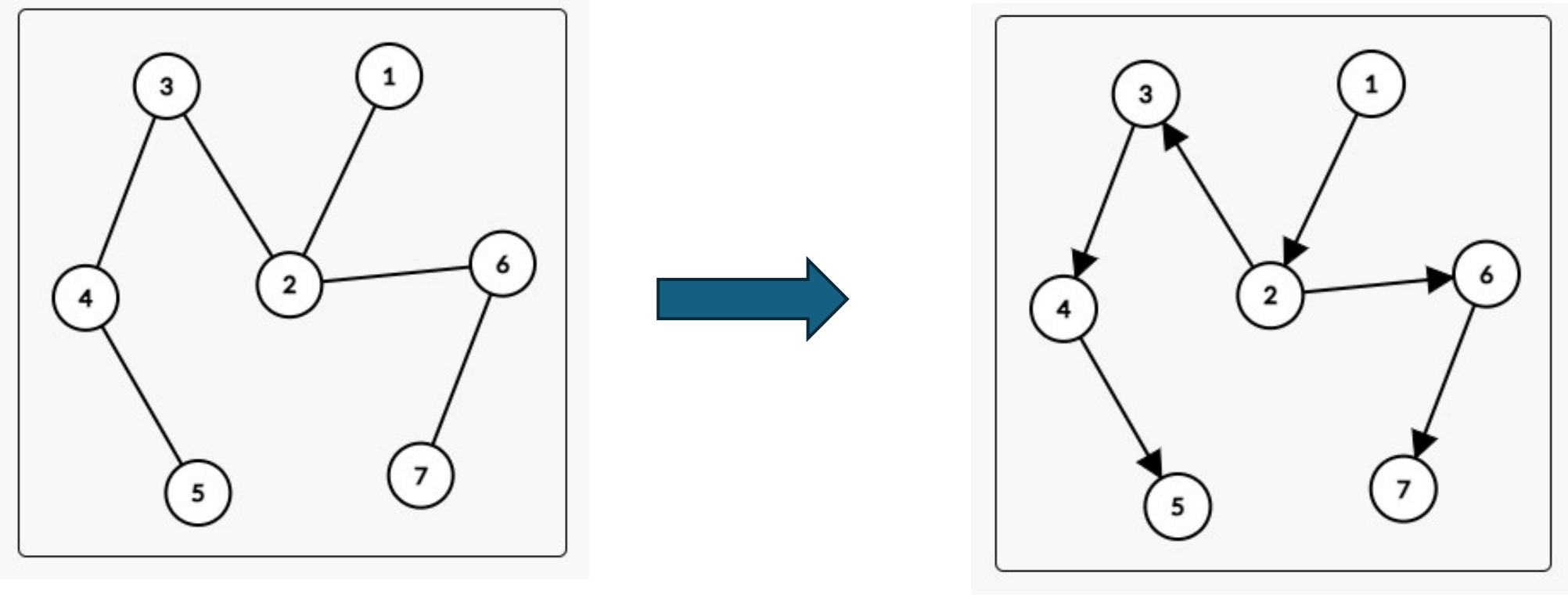
1	1 2
2	2 3
3	4 3
4	4 5
5	6 2
6	6 7



2213: 트리의 독립집합

우선 편의를 위해 양방향 그래프를

1 번 정점을 기준으로 트리가 흐르는 방향으로 그래프를 정리해준다



2213: 트리의 독립집합

g 는 처음에 입력받은 양방향 간선
g[t1].push_back(t2);
g[t2].push_back(t1);
요런식으로 양방향 다 들어가있는 걸

dfs 를 돌며 직접 사용할 단방향 트리 간선으로 바꿔줌
gr[now].push_back(next);

```
void dfs(int now) {
    visited[now] = true;
    for(int next : g[now]) {
        if(visited[next] == false) {
            gr[now].push_back(next);
            dfs(next);
        }
    }
}
```

2213: 트리의 독립집합

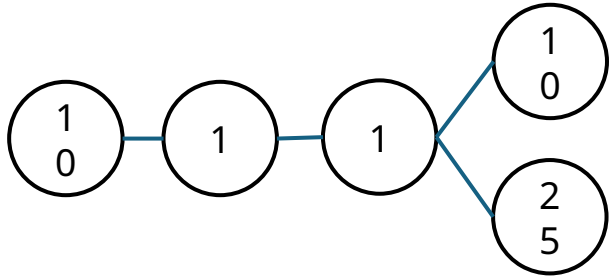
dp[now][state] => 현재 now, state 상태에서 now 포함 트리 아랫부분의 최댓값

now => 현재 노드 번호

state => now 에서 노드를 골랐는지 안 골랐는지
state 에 따라서 다음번 노드를 선택할 수 있는지 없는지 결정됨

0 이라면 다음은 -> 0, 1 두가지 선택지 중 최댓값 1
(두번 연속 안 고르는 경우가 최적인 경우가 있음)

1 이라면 다음은 -> 무조건 0



2213: 트리의 독립집합

가상의 0 번 정점을 만들어서 1 번 정점과 연결해줌 !!!!

그리고 go(0, 0) 으로 시작한다 .

이렇게 되면 go(1, 0), go(1,1) 을 알아서 비교하게 되어 편함

-> gr[0].push_back(1);

2213: 트리의 독립집합

메모이제이션 , 초기화

현재 상태가 0 이라면
다음 노드를 선택하거나
선택하지 않거나 .
둘 중 큰 값을 고르면 됨

1 이라면 무조건 고르지
않는다

```
int go(int now, int state) {
    int &ret = dp[now][state];
    if(ret != -1) {
        return ret;
    }
    ret = 0;

    for(int next : gr[now]) {
        if(state == 0) {
            ret +=  max(a:go(now:next, state:0), b:go(now:next, state:1) + w[next]);
        } else {
            ret += go(now:next, state:0);
        }
    }
    return ret;
}
```

2213: 트리의 독립집합

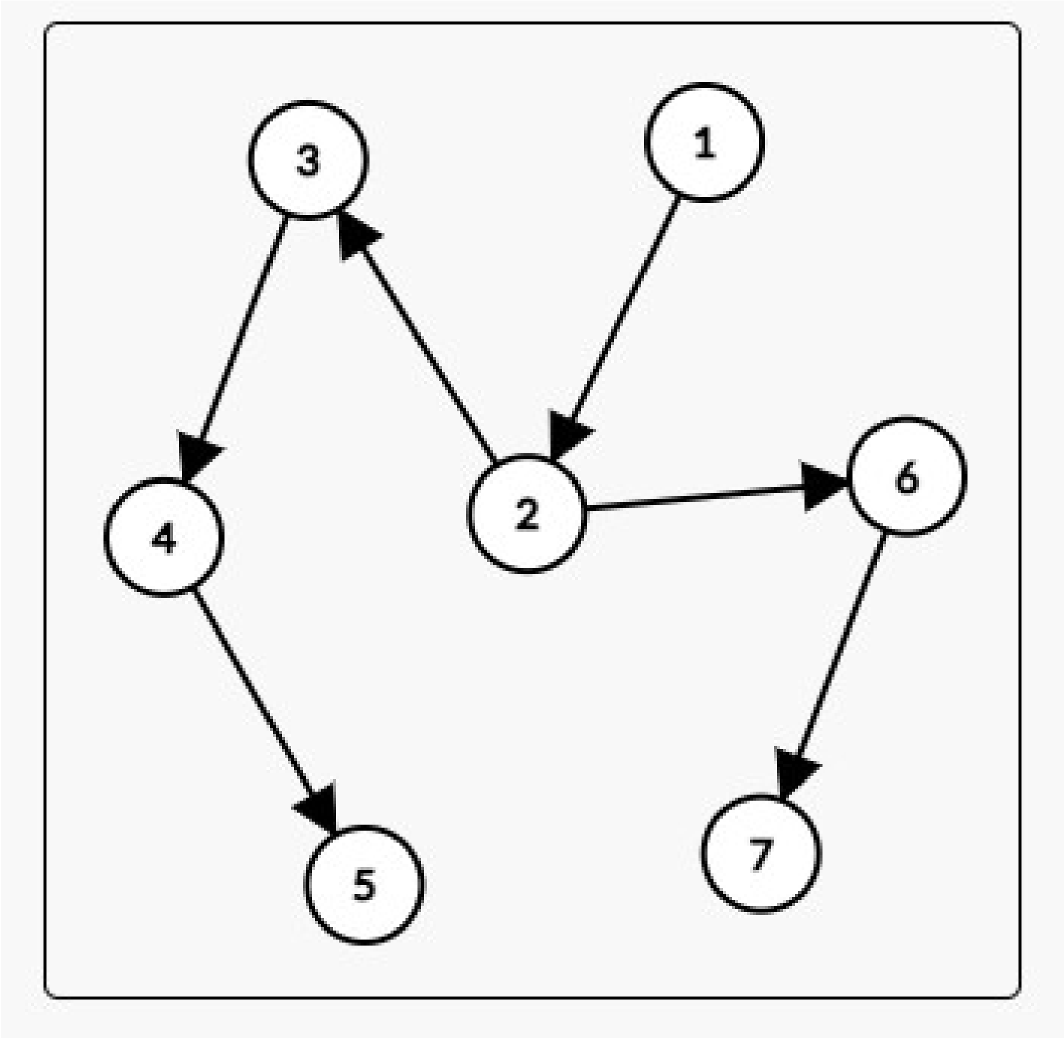
2 번을 고르지 않은 경우

go(2, 0) =
max(go(3, 0), go(3, 1) + w[3]) +
max(go(6, 0), go(6, 1) + w[6])

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\

2 번을 고른 경우

go(2, 1) = go(3, 0) + go(6, 0)



2213: 트리의 독립집합

trace 는 디피와 마찬가지 방법으로
큰거 작은거 비교해서 큰 쪽으로 이동하며

state 가 1 인 경우에 정답 배열에 추가함

```
vector<int> ans;
void trace(int now, int state) {
    if(state == 1) ans.push_back(now);

    for(int next : gr[now]) {
        if(state == 0) {
            if(go(now:next, state:0) < go(now:next, state:1) + w[next]) {
                trace(now:next, state:1);
            } else {
                trace(now:next, state:0);
            }
        } else {
            trace(now:next, state:0);
        }
    }
}
```