#고급반 4주차

동적 계획법: 심화편

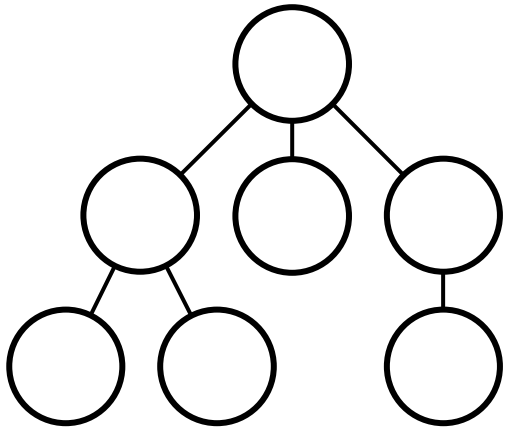
<DP on Tree, DP using Bitmask>

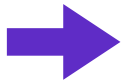
2p /<트리 DP>
19p /<비트마스킹>
26p /<비트 DP>
33p /<연습 문제>

#트리DP

<Dynamic Programming on Tree>

이름에서 유추할 수 있겠지만, 트리에서 DP를 하는 것입니다.
그런데, 그걸 어떻게 할까요?



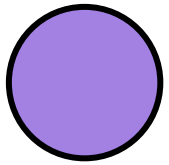


dp[1]	dp[2]	dp[3]	dp[4]	dp[5]	dp[6]	dp[7]

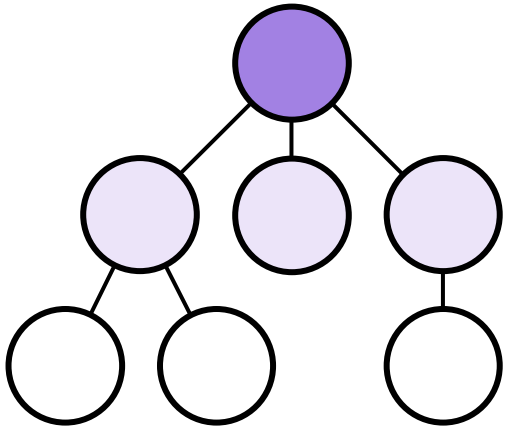
#트리DP

<Dynamic Programming on Tree>

루트가 있는 트리를, 아래와 같이 재귀적으로 생각해볼 수 있습니다.



정점이 1개인 그래프는 트리이다.

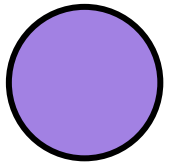


트리만을 자식으로 삼는 그래프는 트리이다.

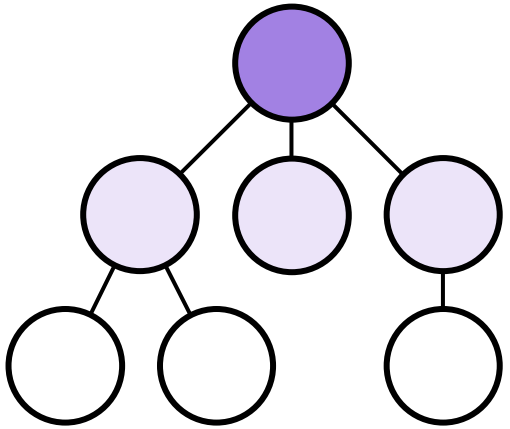
#트리DP

<Dynamic Programming on Tree>

그럼, 아래와 같이 재귀적으로 정의되는 점화식을 생각해볼 수 있습니다.



정점이 1개라면, 초항으로 계산한다.



자식들의 DP값을 기반으로, 점화식을 세워 계산한다.

#트리DP

<Dynamic Programming on Tree>

방금의 재귀적 정의를 조금 더 자세히 분석해봅시다.

1. 초항: 정점이 1개인 경우로, 간단히 계산할 수 있는 경우가 대부분입니다.
2. 점화식: 이전 항들, 즉 자식들의 값을 기반으로 계산합니다.

트리DP라고 해서 원래 DP랑 큰 차이를 가지지는 않습니다.
그저 초항과 점화식이 트리에 맞게 변화하는 것일 뿐이죠.

#트리DP

<Dynamic Programming on Tree> - 구현 틀

오른쪽은 제가 주로 트리DP를 짜는 방식입니다.

각 줄 별로 간단한 설명을 해봅시다.

```
vector<int> adj[200020];

int dp[200020];
void dpf(int now, int pre){
    dp[now]; // = 초항 설정
    for (int nxt : adj[now]){
        if (nxt == pre){ continue; }
        dpf(nxt, now); dp[now]; // = 점화식 계산
    }
}

main(){ dpf(root, -1); }
```

#트리DP

<Dynamic Programming on Tree> - 구현 틀

우선, 초항을 설정합니다.

```cpp
vector<int> adj[200020];

int dp[200020];
void dpf(int now, int pre){
    dp[now]; // = 초항 설정
    for (int nxt : adj[now]){
        if (nxt == pre){ continue; }
        dpf(nxt, now); dp[now]; // = 점화식 계산
    }
}

main(){ dpf(root, -1); }
```

#트리DP

<Dynamic Programming on Tree> - 구현 틀

그 뒤로, 현재 보는 정점과 인접한 정점들을 하나씩 확인합니다.

이 때 pre는 now의 부모 정점으로, 자식들만 확인해야 하는 점화식 상 제외해야 합니다.

```
vector<int> adj[200020];

int dp[200020];
void dpf(int now, int pre){
    dp[now]; // = 초항 설정
    for (int nxt : adj[now]){
        if (nxt == pre){ continue; }
        dpf(nxt, now); dp[now]; // = 점화식 계산
    }
}

main(){ dpf(root, -1); }
```

#트리DP

<Dynamic Programming on Tree> - 구현 틀

이제 남은 건 점화식을 계산하는 것뿐이죠.

```
vector<int> adj[200020];

int dp[200020];
void dpf(int now, int pre){
    dp[now]; // = 초항 설정
    for (int nxt : adj[now]){
        if (nxt == pre){ continue; }
        dpf(nxt, now); dp[now]; // = 점화식 계산
    }
}

main(){ dpf(root, -1); }
```

#트리DP

<Dynamic Programming on Tree> - 연습 문제

트리와 쿼리 (BOJ #15681)

<문제 설명>

- 정점이 N개인 루트 있는 트리 T가 주어집니다.
- 이 때, 아래 쿼리에 Q회 답하세요.
  - v: v를 루트로 하는 서브트리의 정점의 개수를 출력하세요.

<제약 조건>

- $1 \leq N \leq 100\,000$
- $1 \leq Q \leq 100\,000$

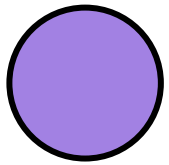
#트리DP

<Dynamic Programming on Tree> - 연습 문제 풀이

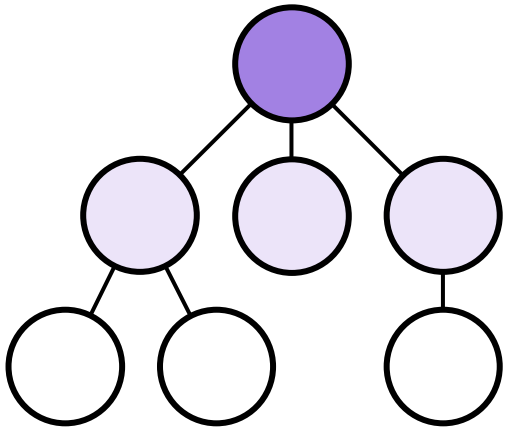
트리와 쿼리 (BOJ #15681)

<문제 해설>

정점 v를 루트로 하는 서브트리의 정점 개수는 아래와 같이 구할 수 있습니다.



정점이 1개라면, 정점 개수는 1개이다.



아닐 경우, 자식들의 정점 개수 합 + 1개이다.

#트리DP

<Dynamic Programming on Tree> - 연습 문제 구현

전체적으로 코드는 아까 본 틀과 비슷합니다.

초항은 아까 본 대로 1개이고,

점화식은 1 + 자식들의 정점 개수 합으로 계산됨을 볼 수 있습니다.

```cpp
vector<int> adj[100020];

int dp[100020];
void dpf(int now, int pre){
    dp[now] = 1;
    for (int nxt : adj[now]){
        if (nxt == pre){ continue; }
        dpf(nxt, now); dp[now] += dp[nxt];
    }
}
```

#트리DP

<Dynamic Programming on Tree> - 연습 문제

사회망 서비스(SNS) (BOJ #2533)

<문제 설명>

- 정점이 N개인 트리가 주어집니다.
- 이 때, 아래 조건을 만족하도록 최소한의 정점을 선택하세요.
  - 모든 정점 v에 대해, v가 선택되어 있거나 v와 인접한 모든 정점이 선택되어 있어야 합니다.

<제약 조건>

- $1 \leq N \leq 1\,000\,000$

#트리DP

<Dynamic Programming on Tree> - 연습 문제 풀이

3 사회망 서비스(SNS) (BOJ #2533)

<문제 해설>

트리 DP를 사용하기에는 뭔가 부적합해 보입니다.

서브트리도 보이지 않고, 심지어는 루트조차 보이지 않습니다.

#트리DP

<Dynamic Programming on Tree> - 연습 문제 풀이

3 사회망 서비스(SNS) (BOJ #2533)

<문제 해설>

하지만 그런 건 만들면 되는 겁니다.

임의로 하나의 정점을 루트로 잡은 뒤, 서브트리에 대한 문제부터 풀어나가봅시다.

문제의 답은 트리 전체, 즉 모든 정점이 포함되는 서브트리에 대해 계산해주면 됩니다.

#트리DP

<Dynamic Programming on Tree> - 연습 문제 풀이

3 사회망 서비스(SNS) (BOJ #2533)

<문제 해설>

그런데 이번에 DP는 어떻게 구해야 할까요?

dp[v][c] = v를 루트로 가지는 서브트리에 대해, 정점 v를 선택했는가 여부를 c로 표현해봅시다.

그럼, 점화식은 c에 대해 나눠서 계산해볼 수 있습니다.

- c = 1: 자기 자신을 선택했으므로, 자신과 인접한 자식들을 마음대로 할 수 있습니다.
- c = 0: 자기 자신을 선택하지 않았으므로, 자신과 인접한 모든 자식들을 선택해야 합니다.

#트리DP

<Dynamic Programming on Tree> - 연습 문제 구현

아까 본 초항과 점화식을 구현한 것입니다.

초항은 자기 자신의 선택 개수를 의미하고,

점화식은 아까 본 대로 해주면 됩니다.

```
vector<int> adj[1000020];
int dp[1000020][2];

void dpf(int now, int pre){
    dp[now][0] = 0; dp[now][1] = 1;
    for (int nxt : adj[now]){
        if (nxt == pre){ continue; }
        dpf(nxt, now);
        dp[now][0] += dp[nxt][1];
        dp[now][1] += min(dp[nxt][0], dp[nxt][1]);
    }
}
```

#비트DP

<Dynamic Programming using Bitfield>

이번에는 비트를 사용한 DP입니다.
하지만 이 전에 비트를 잘 다루기 위해서 필요한 것이 있는데...

#비트마스크

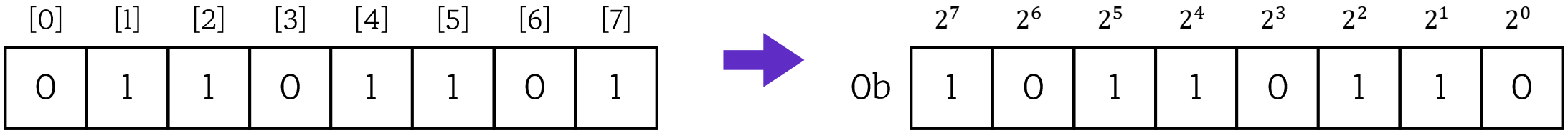
<Bitmask>

바로 비트마스킹입니다.

#비트마스크

<Bitmask>

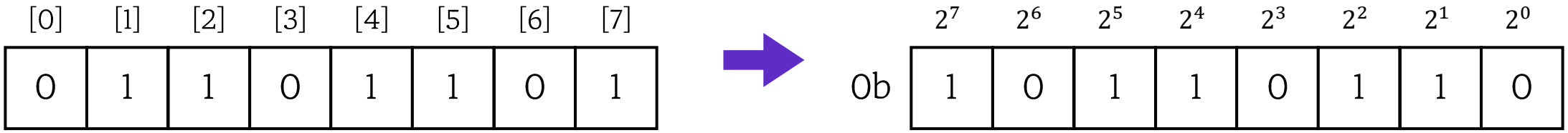
비트마스킹이란, 아래와 같이 하나의 Boolean 배열을
하나의 정수 자료형에 1비트씩 나눠서 저장하는 기법을 의미합니다.



#비트마스크

<Bitmask>

비트마스킹도 결국 Boolean 배열을 의미하므로,
Boolean 배열에서 할 수 있는 연산과 이의 비트마스킹 버전을 봅시다.



#비트마스크

<Bitmask>

Boolean 배열에서 할 수 있는 4가지 연산으로, 위에서부터 각각 원소 접근, 1로 변경, 0으로 변경, 0/1 상태 변환을 의미합니다.

공통적으로, b번째 원소를 접근하기 위해 1 << b를 사용하는 것을 볼 수 있습니다.

```
bool arr[32]; // Boolean 배열
int bit;      // 이의 비트마스킹

// Access on b'th element
arr[b]; bit>>b & 1

// Set b'th element to 1
arr[b] = 1; bit |= 1<<b;

// Set b'th element to 0
arr[b] = 0; bit &= ~(1<<b)

// Toggle b'th element (if 1 then 0 otherwise 1)
arr[b] = !arr[b]; bit ^= 1<<b;
```

#비트마스크

<Bitmask> - 연습 문제

집합 (BOJ #11723)

<문제 설명>

- 공집합 S와, 아래와 같은 Q개의 쿼리가 주어집니다.
  - S에 원소 x를 추가
  - S에서 원소 x를 삭제
  - S에 원소 x가 존재하는지 판별
  - S에 원소 x가 존재한다면 삭제, 아니면 추가
  - S에 1..20을 모두 추가
  - S에서 1..20을 모두 삭제
- Q개의 쿼리를 처리하면 됩니다.

<제약 조건>

- $1 \le Q \le 3\,000\,000$
- $1 \le x \le 20$

#비트마스크

<Bitmask> - 연습 문제 풀이

5 집합 (BOJ #11723)

<문제 해설>

특정 집합에 원소가 존재하는지 아닌지를 비트마스킹으로 저장해봅시다.

초기 상태는 공집합이므로, S = 0000 0000 0000 0000 0000이 됩니다.

그 뒤로는, 아까 본 비트마스킹의 기본 연산들을 구현해주면 됩니다.

#비트마스크

<Bitmask> - 연습 문제 구현

아까 본 비트마스킹의 연산과 동일합니다.

새로 생긴 건 all과 empty 뿐이고, 이는

모든 비트를 1로 세팅한 값을 넣는 방식과

모든 비트를 0으로 세팅한 값을 넣는 방식으로

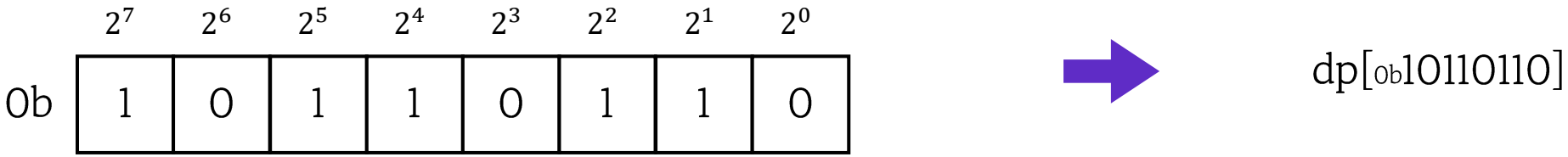
해결할 수 있습니다.

```
int main(){
    int mask = 0;
    int q; cin >> q; while (q--){
        string typ; cin >> typ;
        if (typ == "add"){
            int x; cin >> x;
            mask |= 1<<x;
        }
        if (typ == "remove"){
            int x; cin >> x;
            mask &= ~(1<<x);
        }
        if (typ == "check"){
            int x; cin >> x;
            cout << (mask>>x & 1) << endl;
        }
        if (typ == "toggle"){
            int x; cin >> x;
            mask ^= 1<<x;
        }
        if (typ == "all"){
            mask = 2097150; // 1번부터 20번 비트까지 다 켰을 때의 값
        }
        if (typ == "empty"){
            mask = 0;
        }
    }
}
```

#비트DP

<Dynamic Programming using Bitfield>

이제 비트DP로 돌아와봅시다.
비트DP는, 비트마스킹을 사용한 DP입니다.



#비트DP

<Dynamic Programming using Bitfield> - 연습 문제

외판원 순회 (BOJ #2098)

<문제 설명>

- 정점 N개의 그래프가 주어집니다.
- 모든 정점을 방문하고 시작점으로 돌아오는 가장 짧은 경로의 길이를 출력하세요.

<제약 조건>

- $1 \leq N \leq 16$

#비트DP

<Dynamic Programming using Bitfield> - 연습 문제 풀이

외판원 순회 (BOJ #2098)

<문제 해설>

가장 최근에 방문한 정점을 v라고 하고, 지금까지의 정점 방문 여부를 비트마스킹한 것을 bit라고 합시다.
그럼, 우리는 문제를 dp[v][bit]로 해결할 수 있습니다.

v에서 w로 이동할 때마다, dp[v][bit]의 값을 dp[w][bit | 1<<w]의 업데이트에 써주면 됩니다.
두 인덱스가 가지는 의미는 각각 정점의 이동과 방문 여부의 업데이트를 의미하죠.

#비트DP

<Dynamic Programming using Bitfield> - 연습 문제 구현

오른쪽 코드를 통해 문제를 해결할 수 있습니다.

그런데, 오른쪽 코드를 잘 읽으면

의문을 가질 만한 부분이 많아 보입니다.

```
int adj[20][20];
int dp[20][65540];

int main(){
    // 입력, 0-indexed
    int ful = (1<<n) - 1; // 모든 비트가 켜진 경우
    for (int bit = 0; bit <= ful; bit++){
        for (int now = 0; now < n; now++){
            if (~bit>>now & 1){ continue; }
            if (1<<now == bit){ dp[now][bit] = adj[0][now]; }
            else{ dp[now][bit] = INF; }
            for (int pre = 0; pre < n; pre++){
                if ((pre == now) || (~bit>>pre & 1)){ continue; }
                dp[now][bit] = min(dp[now][bit], dp[pre][bit & ~(1<<now)] +
adj[pre][now]);
            }
        }
    }
    cout << dp[0][ful];
}
```

#비트DP

<Dynamic Programming using Bitfield> - 연습 문제 구현

1. bit를 그냥 이렇게 돌려도 되나요?

네. 점화식 특성상 bit에는 1이 더 켜지는 것일 뿐이므로, 탐색 과정에서 bit는 단조 증가합니다. 그러므로, bit를 증가하는 순서대로 봐줘도 문제 없습니다.

```
int adj[20][20];
int dp[20][65540];

int main(){
    // 입력, 0-indexed
    int ful = (1<<n) - 1; // 모든 비트가 켜진 경우
    for (int bit = 0; bit <= ful; bit++){
        for (int now = 0; now < n; now++){
            if (~bit>>now & 1){ continue; }
            if (1<<now == bit){ dp[now][bit] = adj[0][now]; }
            else{ dp[now][bit] = INF; }
            for (int pre = 0; pre < n; pre++){
                if ((pre == now) || (~bit>>pre & 1)){ continue; }
                dp[now][bit] = min(dp[now][bit], dp[pre][bit & ~(1<<now)] +
adj[pre][now]);
            }
        }
    }
    cout << dp[0][ful];
}
```

#비트DP

<Dynamic Programming using Bitfield> - 연습 문제 구현

2. 초항은 어딨어요?

첫 정점을 방문하는 시점, 즉 방문한 정점이 하나밖에 없는 경우가 초항이 됩니다.

이 때의 값은 0 → v의 거리가 됩니다.

```
int adj[20][20];
int dp[20][65540];

int main(){
    // 입력, 0-indexed
    int ful = (1<<n) - 1; // 모든 비트가 켜진 경우
    for (int bit = 0; bit <= ful; bit++){
        for (int now = 0; now < n; now++){
            if (~bit>>now & 1){ continue; }
            if (1<<now == bit){ dp[now][bit] = adj[0][now]; }
            else{ dp[now][bit] = INF; }
            for (int pre = 0; pre < n; pre++){
                if ((pre == now) || (~bit>>pre & 1)){ continue; }
                dp[now][bit] = min(dp[now][bit], dp[pre][bit & ~(1<<now)] +
adj[pre][now]);
            }
        }
    }
    cout << dp[0][ful];
}
```

#비트DP

<Dynamic Programming using Bitfield> - 연습 문제 구현

3. 점화식은 왜 저래요?

설명할 때는 [v][bit] → [w][bit | 1<<w]였지만,

코드에서는 [v][bit & ~(1<<w)] → [w][bit]를

하고 있습니다.

이 외에는 다른 점이 없습니다.

```
int adj[20][20];
int dp[20][65540];

int main(){
    // 입력, 0-indexed
    int ful = (1<<n) - 1; // 모든 비트가 켜진 경우
    for (int bit = 0; bit <= ful; bit++){
        for (int now = 0; now < n; now++){
            if (~bit>>now & 1){ continue; }
            if (1<<now == bit){ dp[now][bit] = adj[0][now]; }
            else{ dp[now][bit] = INF; }
            for (int pre = 0; pre < n; pre++){
                if ((pre == now) || (~bit>>pre & 1)){ continue; }
                dp[now][bit] = min(dp[now][bit], dp[pre][bit & ~(1<<now)] +
adj[pre][now]);
            }
        }
    }
    cout << dp[0][ful];
}
```

#연습 문제 도전

<Dynamic Programming on Tree>

3 이진 트리 (BOJ #13325)

정점이 아니라 간선에 써볼까요?

2 트리와 수열 (BOJ #26159)

HCPC 기출. (v, w) 간선은 얼마나 중요할까?

5 Tree Product (BOJ #20614)

어떤 이상한 사람이 트리에 곱셈 연산 끼워 넣어요

2 불안정한 물질 (BOJ #18770)

트리도 없는데 왜 트리DP일까요?

1 XOR, Tree, and Queries (BOJ #27492)

무려 Codeforces Div2F번 문제

4 아쉬움이 남지만 (BOJ #18862)

내고 싶은 거 많았는데 6개밖에 못 내서 아쉬워하는
학술부장의 마음을 담아서

#연습 문제 도전

<Dynamic Programming using Bitfield>

5 두 단계 최단 경로 4 (BOJ #23480)

정점은 많은데 방문해야 하는 정점은 적은 문제

5 박성원 (BOJ #1086)

박성원은 이 문제를 풀지 못했다.

4 Uddered but not herd (BOJ #20967)

제한이 신기한 문제

5 발전소 (BOJ #1102)

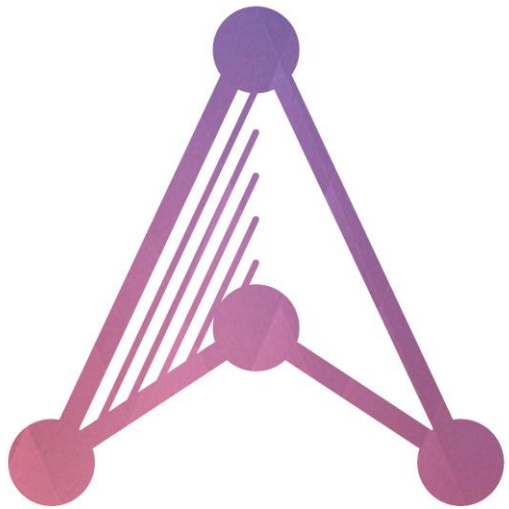
모두 고칠 필요 없고 P개만 고치면 되는 문제

4 대충 정렬 (BOJ #4800)

아 몰라 설명 대충 적을게요

3 격자판 채우기 (BOJ #1648)

M = 2면 얼마나 편했을까



ALOHA

The algorithm club.