A. 나무를 심어볼까요?

1. 나무 심기는 왼쪽에서 오른쪽 방향으로 진행한다.
2. 비어있는 칸 중 가장 왼쪽 칸에 나무를 심었을 때 이 나무가 햇빛을 받을 수 있다면, 바로 나무를 심는다.
3. 만약 나무를 심었을 때 이 나무가 햇빛을 받을 수 없다면, 조건에 따라 다음 중 하나를 수행한다.
    - 현재 심어져 있는 나무 중 가장 오른쪽의 나무가 아직 심어진 지 K일이 되지 않았다면, 그 나무를 뽑아서 버린 후, 1의 과정으로 돌아간다.
    - 현재 심어져 있는 나무 중 가장 오른쪽의 나무가 심어진 지 K일 이상이 되었다면, 이미 뿌리가 깊게 박혀서 뽑기 힘들기 때문에 한 칸 띄워서 나무를 심는다. 이때 생기는 빈 칸에는 다시 나무를 심을 수 없다.

- 하루에 나무를 하나씩 무조건 심는 조건 + 특정 조건 만족 시 [1의 과정으로 돌아가는 회귀류 문제]

- 변량 : 높이, 시간, 위치

- 풀이 : 심은 나무(struct tree : h, t, p)를 스택에 넣은 다음, 회귀할때마다 조건 비교 후 빼내고 넣으면서 풀이

- Q. 스택 아니고 좀 더 깔끔한 풀이가 존재할지

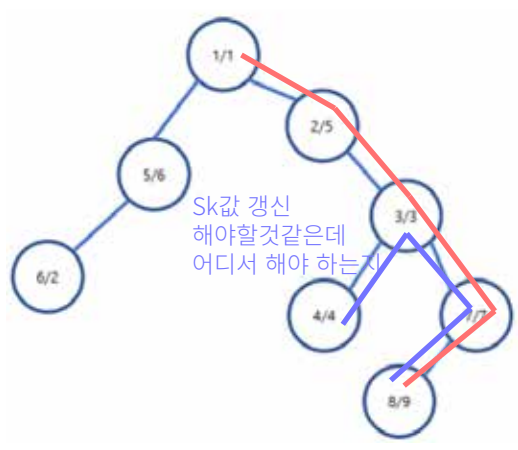
- 카카오 공채문제 유형과 비슷함(괄호 변환)

```
1. 입력이 빈 문자열인 경우, 빈 문자열을 반환합니다.
2. 문자열 w를 두 "균형잡힌 괄호 문자열" u, v로 분리합니다. 단, u는 "균형잡힌 괄호 문자열"로 더 이
3. 문자열 u가 "올바른 괄호 문자열" 이라면 문자열 v에 대해 1단계부터 다시 수행합니다.
  3-1. 수행한 결과 문자열을 u에 이어 붙인 후 반환합니다.
4. 문자열 u가 "올바른 괄호 문자열"이 아니라면 아래 과정을 수행합니다.
  4-1. 빈 문자열에 첫 번째 문자로 '('를 붙입니다.
  4-2. 문자열 v에 대해 1단계부터 재귀적으로 수행한 결과 문자열을 이어 붙입니다.
  4-3. ')'를 다시 붙입니다.
  4-4. u의 첫 번째와 마지막 문자를 제거하고, 나머지 문자열의 괄호 방향을 뒤집어서 뒤에 붙입니다.
  4-5. 생성된 문자열을 반환합니다.
```

F. 바이토닉 트리

```cpp
#include<iostream>

using namespace std;
int n, r;
int node[100005];

struct edge {
    int s, t, prv;
} e[100005];
int l[100005];

bool isnot;

// (처음 노드 번호, Sk값, 증가 감소 상태(증가 : true, 감소 : false)
void dfs(int srt, int max, bool stat) {
    for (int i = l[srt]; i; i = e[i].prv) {
        // 간선의 시작 노드값과 끝점 노드값
        int vs = node[e[i].s];
        int vt = node[e[i].t];

        // 감소상태인데 Sk < Sn-1 인 경우 리턴한다
        if (stat == false && max < vt) {
            isnot = true; return;
        }
        // 증가상태라면 Sk값 갱신하고 자식 노드 탐색
        if (vs < vt) {
            max = vt;
            dfs(e[i].t, max, true);
        }
        // 감소상태라면 감소상태로 갱신(false)후 다음 노드 탐색
        else {
            dfs(e[i].t, max, false);
        }

        // Q. 리프 노드까지 탐색 후 다른 가지로 들어갈 때
        // 수열의 Sk값을 어디서 갱신해야 하는지 모르겠습니다.
    }
}
```

```cpp
int main() {
    cin >> n >> r;

    for (int i = 0; i < n; i++) {
        cin >> node[i + 1];
    }
    // 수업 배운대로 adjList
    for (int i = 1; i < n; i++) {
        cin >> e[i].s >> e[i].t;
        e[i].prv = l[e[i].s];
        l[e[i].s] = i;
    }
    // adjList를 통한 tree traverse
    dfs(r, node[r], true);

    if (isnot) {
        cout << "No" << '₩n';
    }
    else {
        cout << "Yes" << '₩n';
    }
    return 0;
}
```



D. 숭고한 캠프

- 처음엔 Trie 구조를 만들고 so각 단어의 counting이 두번 이상 되는 단어는 빼는 식으로 생각 (필요성을 못느껴 버림)

- 각 동아리 단어들 중 다른 동아리 단어와 구분되는(Unique) 단어의 조합을 만들고, 그 조합의 개수를 세는 것

- 각 동아리의 단어들을 set을 통하여 중복 없이 check에 넣음 (check[word-'A']++)

- 다시 동아리 단어들을 순회하며 check[word-'A'] == 1인 것만 뽑아냄

- Q. But. 뽑아낸 단어들을 다시 조합하는 방법이 떠오르지 않음...

G. 트리 가짓수 세기

- BST에서 insert 시 BST의 높이를 K로 제어하는 방법 (떠오르지 않음)

- Q. BST insert 코드만 외워서 사용중입니다.
  Trie, adjList, LinkedList, Heap과 같이 더 효율적인 코드가 있는지 궁금합니다.
  있다면 배워서 쓰겠습니다.

- 사용중인 BST Code

```cpp
struct info {
    int x, y;
    int count;
};
class node {
public:
    info data;
    node* left, *right;
    node() : data({ 0,0,0 }), left(nullptr), right(nullptr) {};
    node(info key) : data(key), left(nullptr), right(nullptr) {};
};
```

```cpp
class Tree {
public:
    node* root, *current, *parent;
    Tree() : root(nullptr), current(nullptr), parent(nullptr) {};

    void insert(node* target) {
        if (root == nullptr) {
            root = target;
            return;
        }
        else {
            current = parent = root;
            if (!search(target->data)) {
                if (current->data.x < target->data.x) {
                    current->right = target;
                }
                else {
                    current->left = target;
                }
            }
        }
    }

    node* search(info key) {
        current = parent = root;
        while (current) {
            if (current->data.x == key.x) return current;
            parent = current;
            if (current->data.x < key.x) {
                current = current->right;
            }
            else {
                current = current->left;
            }
        }
        return nullptr;
    }
};
```

BOJ 2842. 집배원 한상덕

BOJ 10849. A Journey to Greece

문제만 읽어보고 가겠습니다..ㅎ 추후 복습할게요