2022-2 HI-ARC 중급스터디

4주차. 트리

이지은 (leeju1013)

목차

1. 트리
   – 11725. 트리의 부모 찾기
   – 3584. 가장 가까운 공통 조상

2. 이진 트리
   – 1991. 트리 순회

3. 힙

트리의 부모 찾기 성공

실버 II

시간 제한	메모리 제한	제출	정답
1 초	256 MB	45212	19628

문제

루트 없는 트리가 주어진다. 이때, 트리의 루트를 1이라고 정했을 때, 각 노드의 부모를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 노드의 개수 N (2 ≤ N ≤ 100,000)이 주어진다. 둘째 줄부터 N-1개의 줄에 트리 상에서 연결된 두 정점이 주어진다.

출력

첫째 줄부터 N-1개의 줄에 각 노드의 부모 노드 번호를 2번 노드부터 순서대로 출력한다.

예제 입력 1 복사

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

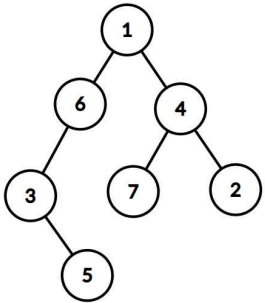
예제 출력 1 복사

```
4
6
1
3
1
4
```

1. 트리

- What?

- 무방향 비순환 연결 그래프



- 사이클이 없음
  : 임의의 서로 다른 두 정점을 잇는 경로가 유일함

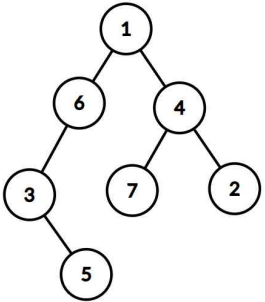
- 모든 정점(노드)이 이어져있음
  : 정점 개수 N개이면 간선 개수 N-1개

- 계층형 구조

1. 트리

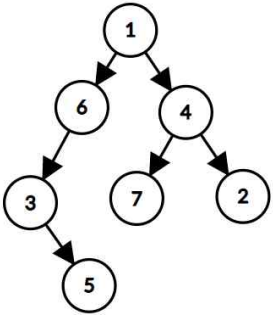
- What?

 - 무방향 비순환 연결 그래프



- 부모-자식 관계
  : 상위-하위 노드

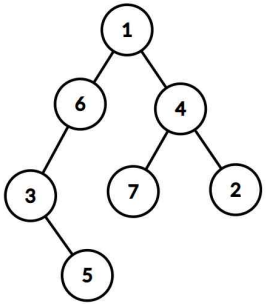
- 루트노드
  : 부모가 없는 노드



1. 트리

- What?

- 무방향 비순환 연결 그래프



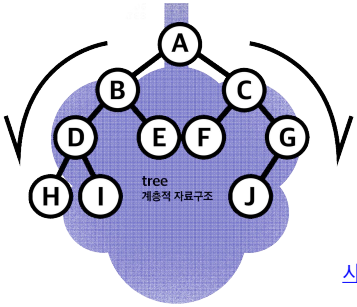
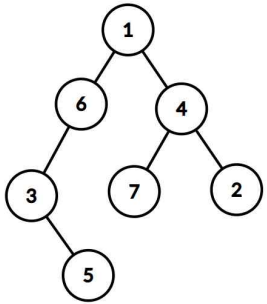


사진 출처

1. 트리

- What?

- 무방향 비순환 연결 그래프



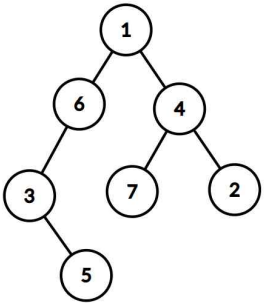
- 서브 트리(subtree)
  : 한 노드의 자식을 루트로 하는 트리

- 리프 노드(leaf node)
  : 자식이 없는 노드

1. 트리

- What?

 - 무방향 비순환 연결 그래프



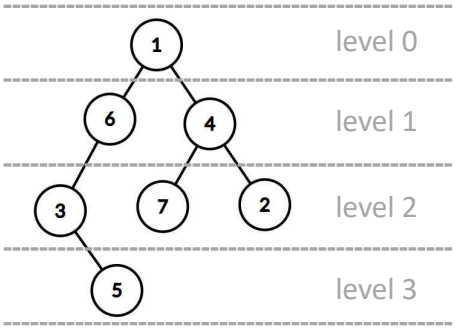
- 형제(sibling)
  : 부모 노드가 같은 노드들

- 조상(ancestors)
  : 부모 노드들의 집합

1. 트리

- What?

- 무방향 비순환 연결 그래프



- 노드의 레벨(level) / 깊이(depth)
  : 루트 노드와의 거리

- 트리의 높이(height)
  : 가장 깊은 정점의 깊이

1. 트리

- How?

인접 행렬

+ 간선 여부를 O(1)에 확인

- 공간 복잡도 O(|V^2|)

-> 간선 수가 많은 경우 사용

	1	2	3	4	5	6	7
1	0	0	0	1	0	1	0
2	0	0	0	1	0	0	0
3	0	0	0	0	1	1	0
4	1	1	0	0	0	0	1
5	0	0	1	0	0	0	0
6	1	0	1	0	0	0	0
7	0	0	0	1	0	0	0

인접 리스트

+ 공간 복잡도 O(|V|+|E|)

- 간선 여부를 O(degree)에 확인

-> 정점 수가 많은 경우 사용

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

* 간선의 수가 정점의 수 -1 뿐이므로 인접 리스트를 쓰는 것이 유리함!

문제

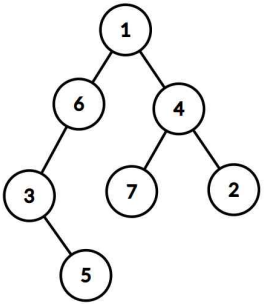
루트 없는 트리가 주어진다. 이때, 트리의 루트를 1이라고 정했을 때, 각 노드의 부모를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 노드의 개수 N (2 ≤ N ≤ 100,000)이 주어진다. 둘째 줄부터 N-1개의 줄에 트리 상에서 연결된 두 정점이 주어진다.

출력

첫째 줄부터 N-1개의 줄에 각 노드의 부모 노드 번호를 2번 노드부터 순서대로 출력한다.



예제 입력 1 복사

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

예제 출력 1

```
4
6
1
3
1
4
```

```cpp
#include <iostream>
#include <vector>
using namespace std;

#define NMAX 100000
vector<int> adj[NMAX+1];
int parent[NMAX+1]; // parent[a]=b ;b가 a의 부모이다(a의 부모는 b이다)

void dfs(int cur){
    for(auto &next :adj[cur]){
        if(parent[cur]!=next){ //나랑 연결된 노드중에 부모가 아닌 노드들
            parent[next]=cur; //의 부모를 나로 설정해줌
            dfs(next);
        }
    }
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int n,a,b; cin>>n;
    for(int i=0; i<n-1; i++){
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    dfs(1); //트리의 루트부터 탐색 시작

    for(int i=2; i<=n; i++){
        cout<<parent[i]<<'\n';
    }
    return 0;
}
```

문제

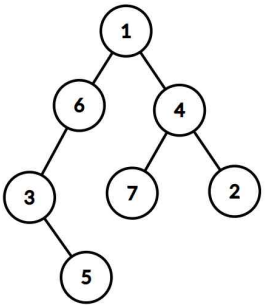
루트 없는 트리가 주어진다. 이때, 트리의 루트를 1이라고 정했을 때, 각 노드의 부모를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 노드의 개수 N (2 ≤ N ≤ 100,000)이 주어진다. 둘째 줄부터 N-1개의 줄에 트리 상에서 연결된 두 정점이 주어진다.

출력

첫째 줄부터 N-1개의 줄에 각 노드의 부모 노드 번호를 2번 노드부터 순서대로 출력한다.



예제 입력 1 복사

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

예제 출력 1

```
4
6
1
3
1
4
```

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

#define NMAX 100000
vector<int> adj[NMAX+1];
int parent[NMAX+1]; // parent[a]=b ;b가 a의 부모이다(a의 부모는 b이다)
int visited[NMAX+1];

void bfs(int root){
    queue<int> q;
    q.push(root);
    visited[root]=1;
    while(!q.empty()){
        int cur = q.front(); q.pop();
        for(int next :adj[cur]){
            if(!visited[next]){
                visited[next]=1;
                parent[next]=cur;
                q.push(next);
            }
        }
    }
}
```

```cpp
void bfs(int root){
    queue<int> q;
    q.push(root);
    while(!q.empty()){
        int cur = q.front(); q.pop();
        for(int next :adj[cur]){
            if(parent[cur]!=next){
                parent[next]=cur;
                q.push(next);
            }
        }
    }
}
```

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int n,a,b; cin>>n;
    for(int i=0; i<n-1; i++){
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    bfs(1); //트리의 루트부터 탐색 시작

    for(int i=2; i<=n; i++){
        cout<<parent[i]<<'\n';
    }
    return 0;
}
```

가장 가까운 공통 조상 상급 다국어

☆ 한국어

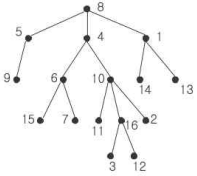
골드 IV

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	6129	3165	2375	52.848%

문제

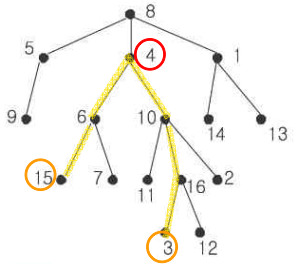
루트가 있는 트리(rooted tree)가 주어지고, 그 트리 상의 두 정점이 주어질 때 그들의 가장 가까운 공통 조상(Nearest Common Anscestor)은 다음과 같이 정의됩니다.

- 두 노드의 가장 가까운 공통 조상은, 두 노드를 모두 자손으로 가지면서 깊이가 가장 깊은(즉 두 노드에 가장 가까운) 노드를 말합니다.



예를 들어 15와 11를 모두 자손으로 갖는 노드는 4와 8이 있지만, 그 중 깊이가 가장 깊은(15와 11에 가장 가까운) 노드는 4 이므로 가장 가까운 공통 조상은 4가 됩니다.

루트가 있는 트리가 주어지고, 두 노드가 주어질 때 그 두 노드의 가장 가까운 공통 조상을 찾는 프로그램을 작성하세요



루트가 있는 트리가 주어짐,
두 노드가 주어질 때
그 두 노드의 가장 가까운 공통 조상을 출력하자.

*최소 공통 조상(LCA, Lowest Common Ancestor)

15와 3의 가장 가까운 공통 조상은 4

첫 줄에 테스트 케이스의 개수 T가 주어집니다.

각 테스트 케이스마다, 첫째 줄에 트리를 구성하는 노드의 수 N이 주어집니다. (2 ≤ N ≤ 10,000)

그리고 그 다음 N-1개의 줄에 트리를 구성하는 간선 정보가 주어집니다. 한 간선 당 한 줄에 두 개의 숫자 A B 가 순서대로 주어지는데, 이는 A가 B의 부모라는 뜻입니다. (당연히 정점이 N개인 트리는 항상 N-1개의 간선으로 이루어집니다!) A와 B는 1 이상 N 이하의 정수로 이름 붙여집니다.

테스트 케이스의 마지막 줄에 가장 가까운 공통 조상을 구할 두 노드가 주어집니다.

출력

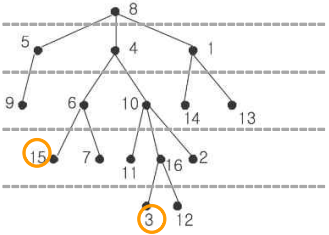
각 테스트 케이스 별로, 첫 줄에 입력에서 주어진 두 노드의 가장 가까운 공통 조상을 출력합니다.

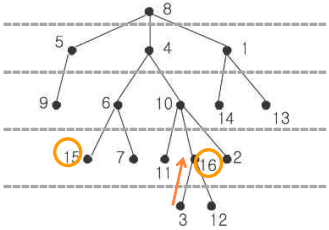
예제 입력 1 복사

```
2
16
1 14
8 5
10 16
5 9
4 6
8 4
4 10
1 13
6 15
10 11
6 7
10 2
16 3
8 1
16 12
16 7
5
2 3
3 4
3 1
1 5
3 5
```

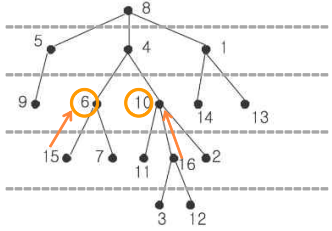
예제 출력 1

```
4
3
```

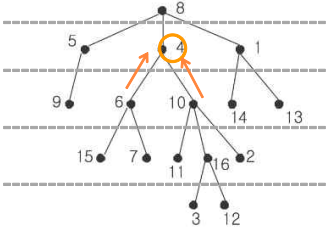




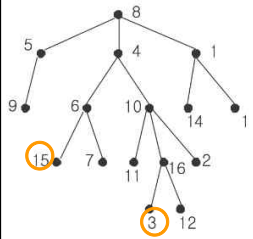
두 정점의 레벨이
같아질 때까지
아래쪽 정점을
위로 올리자

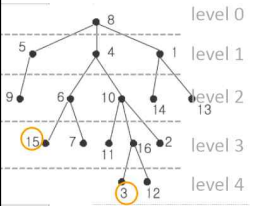


만날 때까지
둘 다
한 칸씩
위로 올리자

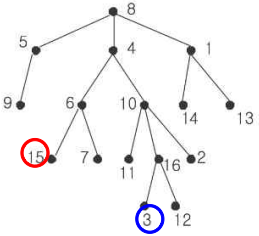


만났다!
LCA 발견!!

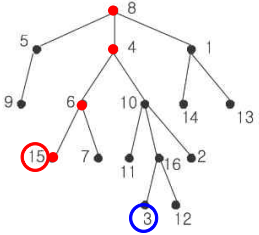




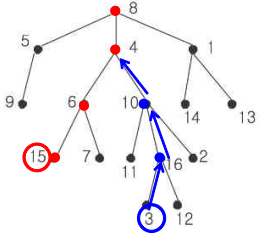

```cpp
#include <iostream>
#include <vector>
using namespace std;
#define NMAX 10000
vector <int> child[NMAX+1];
int depth[NMAX+1];
int parent[NMAX+1];

void checkdepth(int idx, int level){
    depth[idx] = level;
    for (auto &k:child[idx])
        checkdepth(k, level+1);
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    int tc; cin >> tc;
    while(tc--){
        int n,a,b; cin >> n;

        //전역 변수 초기화
        for (auto &k:child)
            k.clear();
        fill(depth, depth+NMAX+1, 0);
        fill(parent, parent+NMAX+1, 0);

        //트리 입력받기
        for (int i=0; i<n-1; i++){
            cin >> a >> b; //a가 b의 부모
            child[a].push_back(b);
            parent[b] = a;
        }

        //root 찾기
        int root;
        for (root=1; root<=n; root++)
            if (!parent[root]) //부모가 없으면 루트임
                break;

        checkdepth(root, 0); //노드들의 depth 계산&저장

        cin >> a >> b; //LCA를 구하려는 두 노드 입력받기

        //두 정점의 레벨이 같아질 때까지 아래쪽 정점을 위로 올리기
        while(depth[a] < depth[b]) b = parent[b];
        while(depth[a] > depth[b]) a = parent[a];

        //두 정점 만날 때까지 사이좋게 위로 올리기
        while(a != b) {
            b = parent[b];
            a = parent[a];
        }

        //LCA 출력!
        cout << a << '\n';
    }
}
```



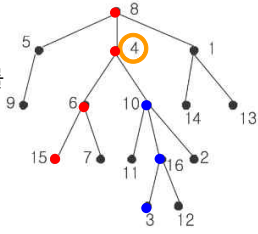
다른 풀이!



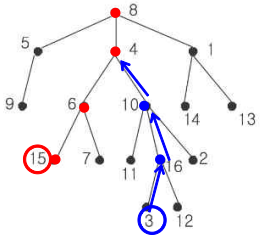
한 노드에서
루트 노드까지 올라가며
방문 여부를 기록하자



다른 노드에서는
빨간 노드의 방문 여부를
확인하면서
루트로 올라가자



파란 노드가 올라가면서
가장 먼저 발견한
방문 흔적이
LCA !




```cpp
#include <iostream>
using namespace std;
#define NMAX 10000
int parent[NMAX+1];
bool visit[NMAX+1];

int main(){
    ios_base::sync_with_stdio(0); cin.tie(0);

    int tc; cin >> tc;
    while(tc--){
        int n, a, b; cin >> n;

        //전역 변수 초기화
        fill(parent, parent+NMAX+1, 0);
        fill(visit, visit+NMAX+1, 0);

        //트리 입력받기
        for (int i=0; i<n-1; i++){
            cin >> a >> b; //a가 b의 부모
            parent[b] = a;
        }

        cin >> a >> b; //LCA를 구하려는 두 노드 입력받기

        while(a != 0){ //a에서 루트까지 올라가며 방문 기록
            visit[a] = 1;
            a = parent[a];
        }
        while(!visit[b]) //a가 방문했던 노드를 처음으로 만날때까지 b를 올리기
            b = parent[b];

        cout << b << '\n'; //LCA 출력!
    }
    return 0;
}
```

쉬는 시간

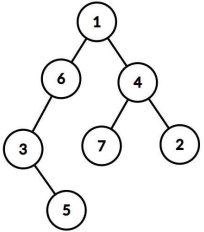
1. 트리
   - 11725. 트리의 부모 찾기
   - 3584. 가장 가까운 공통 조상

2. 이진 트리
   - 1991. 트리 순회

3. 힙

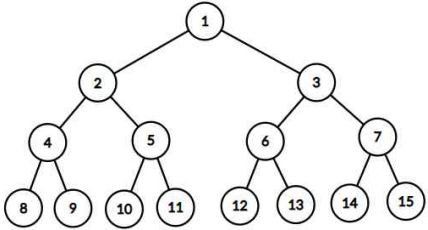
2. 이진 트리 (Binary tree)

- What?
  - 모든 정점이 2개 이하의 자식을 가지는 트리



높이 h인 이진트리의
최대 노드 수 2^h-1개
최소 높이 ceil(logN)

높이 4인 이진트리의
최대 노드 수 15개
최소 높이 4

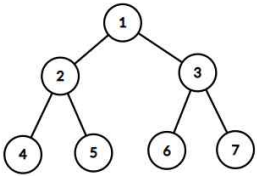


2. 이진 트리 (Binary tree)

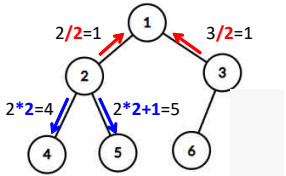
- What?

  - 모든 정점이 2개 이하의 자식을 가지는 트리

- 포화 이진 트리(Full binary tree)
  : 리프노드를 제외한 모든 정점의 자식이 2개인 트리

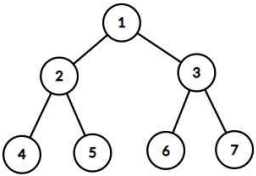


- 완전 이진 트리(Complete binary tree)
  : 마지막 레벨을 제외 하고 모든 레벨이 완전히 채워져 있음,
  마지막 레벨은 왼쪽부터 채워짐



2. 이진 트리 (Binary tree)

- 이진 트리 순회



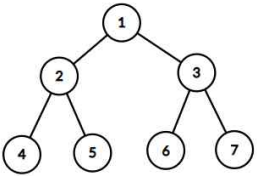
1. 전위 순회 (Preorder)
2. 중위 순회 (Inorder)
3. 후위 순회 (Postorder)

DFS

4. 레벨 순회 (Level-order) : BFS

2. 이진 트리 (Binary tree)

- 이진 트리 순회



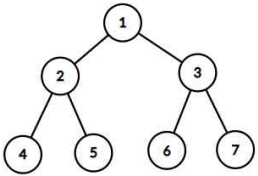
1. 전위 순회 (Preorder)
   : VLR (Visit - Left - Right)

```
1 void preorder(int par){
2    cout<< par << " ";
3    if(left_child[par] != 0) preorder(left_child[par]);
4    if(right_child[par] != 0) preorder(right_child[par]);
5 }
```

출력 : 1 2 4 5 3 6 7

2. 이진 트리 (Binary tree)

- 이진 트리 순회



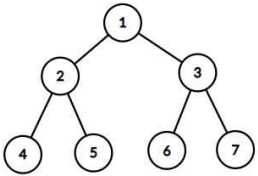
2. 중위 순회 (Inorder)
: LVR (Left - Visit - Right)

```
1 void inorder(int par){
2    if(left_child[par] != 0) preorder(left_child[par]);
3    cout<< par << " ";
4    if(right_child[par] != 0) preorder(right_child[par]);
5 }
```

출력 : 4 2 5 1 6 3 7

2. 이진 트리 (Binary tree)

- 이진 트리 순회



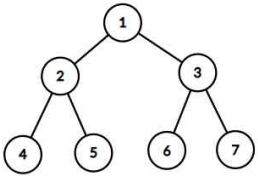
3. 후위 순회 (Postorder)
: LRV (Left - Right - Visit)

```
1 void postorder(int par){
2    if(left_child[par] != 0) preorder(left_child[par]);
3    if(right_child[par] != 0) preorder(right_child[par]);
4    cout<< par << " ";
5 }
```

출력 : 4 5 2 6 7 3 1

2. 이진 트리 (Binary tree)

- 이진 트리 순회



4. 레벨 순회 (Level-order)
: 낮은 레벨의 노드들부터 순서대로 순회

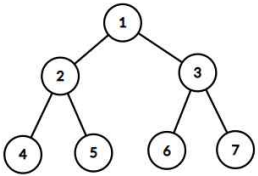
```
1 queue<int> q;
2 q.push(root);
3 while(!q.empty()){
4     int cur = q.front(); q.pop();
5     cout << cur;
6     for(int &next : child[cur])
7         q.push(next);
8 }
```

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

출력 : 1 2 3 4 5 6 7

2. 이진 트리 (Binary tree)

- 이진 트리 순회



1. 전위 순회 (Preorder) : 1 2 4 5 3 6 7
2. 중위 순회 (Inorder) : 4 2 5 1 6 3 7
3. 후위 순회 (Postorder) : 4 5 2 6 7 3 1
4. 레벨 순회 (Level-order) : 1 2 3 4 5 6 7

트리 순회

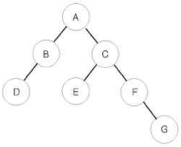
실금

실버 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	37069	24008	18288	65.926%

문제

이진 트리를 입력받아 전위 순회(preorder traversal), 중위 순회(inorder traversal), 후위 순회(postorder traversal)한 결과를 출력하는 프로그램을 작성하시오.



입력

첫째 줄에는 이진 트리의 노드의 개수 N(1 ≤ N ≤ 26)이 주어진다. 둘째 줄부터 N개의 줄에 걸쳐 각 노드와 그의 왼쪽 자식 노드, 오른쪽 자식 노드가 주어진다. 노드의 이름은 A부터 차례대로 알파벳 대문자로 매겨지며, 항상 A가 루트 노드가 된다. 자식 노드가 없는 경우에는 .으로 표현한다.

출력

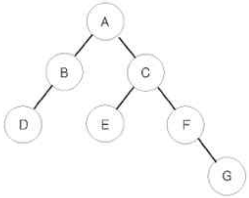
첫째 줄에 전위 순회, 둘째 줄에 중위 순회, 셋째 줄에 후위 순회한 결과를 출력한다. 각 줄에 N개의 알파벳을 공백 없이 출력하면 된다.

예제 입력 1 복사

```
7
A B C
B D .
C E F
E . .
F . G
D . .
G . .
```

예제 출력 1 복사

```
ABDCEFG
DBAECFG
DBEGFCA
```



예제 입력 1 복사

```
7
A B C
B D .
C E F
E . .
F . G
D . .
G . .
```

예제 출력 1 복사

```
ABDCEFG
DBAECFG
DBEGFCA
```

```cpp
#include <iostream>
using namespace std;

int childL[26], childR[26];

void preorder(int idx){
   cout<< char(idx+'A');
   if(childL[idx] != '.'-'A') preorder(childL[idx]);
   if(childR[idx] != '.'-'A') preorder(childR[idx]);
}
void inorder(int idx){
    if(childL[idx] != '.'-'A') inorder(childL[idx]);
    cout<< char(idx+'A');
    if(childR[idx] != '.'-'A') inorder(childR[idx]);
}
void postorder(int idx){
    if(childL[idx] != '.'-'A') postorder(childL[idx]);
    if(childR[idx] != '.'-'A') postorder(childR[idx]);
    cout<< char(idx+'A');
}
```

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int n; cin>>n;
    while(n--){
        char a,b,c; cin>>a>>b>>c;
        childL[a-'A'] = b-'A';
        childR[a-'A'] = c-'A';
    }
    preorder(0); cout<<'\n';
    inorder(0); cout<<'\n';
    postorder(0); cout<<'\n';
    return 0;
}
```

3. 힙(Heap)

- What?

- 최댓값이나 최솟값을 빠르게 찾아내도록 만들어진 자료구조

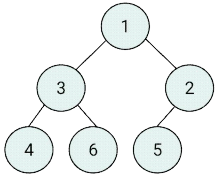
- 완전 이진 트리 + heap property

- 루트 노드의 우선순위 > 해당 서브트리 노드들의 우선순위

- 우선순위 큐 구현에 사용되는 자료구조

3. 힙(Heap)

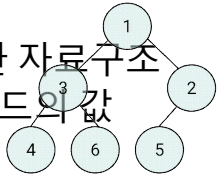
- 최소 힙(Min heap)
  - 최솟값을 빠르게 찾기위한 자료구조
  - 부모 노드의 값 <= 자식 노드의 값



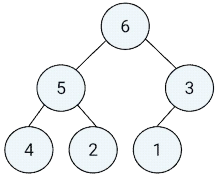
Min heap

사진 출처

- 최대 힙(Max heap)
  - 최댓값을 빠르게 찾기위한 자료구조
  - 부모 노드의 값 >= 자식 노드의 값



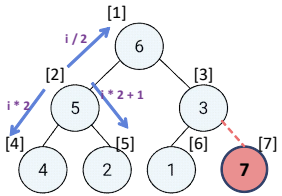
Min heap



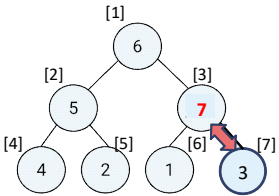
Max Heap

3. 힙(Heap)

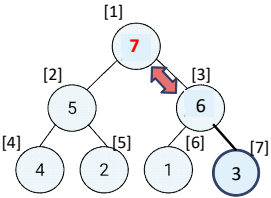
- 삽입(Insertion)



Max Heap



Max Heap



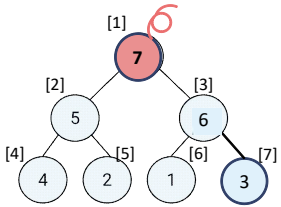
Max Heap

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

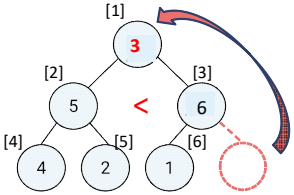
[1]	[2]	[3]	[4]	[5]	[6]	[7]
7	5	6	4	2	1	3

3. 힙(Heap)

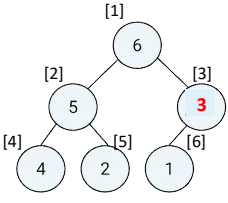
- 삭제(Deletion)



Max Heap



Max Heap



Max Heap

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

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

3. 힙(Heap)

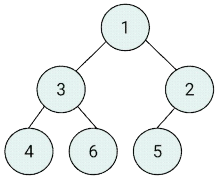
- 시간 복잡도?

- 트리의 높이 : O(logN)에 비례 (∵완전 이진 트리)

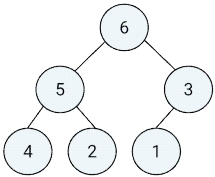
- 삽입(Insertion) : O(logN)

- 삭제(Deletion) : O(logN)

- 접근(Top access) : O(1)



Min heap



Max Heap

3. 힙(Heap)

- What?

- 최댓값이나 최솟값을 빠르게 찾아내도록 만들어진 자료구조

- 완전 이진 트리 + heap property

- 루트 노드의 우선순위 > 해당 서브트리 노드들의 우선순위

- **우선순위 큐** 구현에 사용되는 자료구조

감사합니다

- 필수 문제
  11725. 트리의 부모 찾기
  1991. 트리 순회
- 연습 문제
  3584. 가장 가까운 공통 조상
  2644. 촌수계산
  14675. 단절점과 단절선
  14267. 회사 문화 1
  15681. 트리와 쿼리
  16437. 양 구출 작전
  1967. 트리의 지름
  9934. 완전 이진 트리
  4256. 트리
- 11월 2일(수요일) 저녁 6시 T동