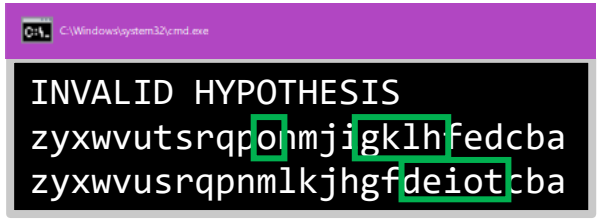
Problem 1:
무작위로 알파벳 순서가 섞인 암호화 사전

- 범죄에 연루된 사람들의 암호화된 인명부 사전이 수사 중 발견되었다.
  - 암호화된 인명부에 포함된 단어들은 모두 영어의 **소문자** 알파벳으로 구성되어 있지만, 사전에 포함된 단어의 순서들이 영어와 서로 달랐다.
  - 수사팀은 단어들이 사전순이 아닌 다른 순서대로 정렬되어 있는지, 아니면 알파벳들의 순서가 영어와 서로 다른 것인지를 알고 싶어한다.
  - 수사팀은 이 암호화된 사전에서는 알파벳들의 순서가 영어와 서로 다를 뿐, 사전의 단어들은 사전 순서대로 배치되어 있다는 가설을 세웠다.
  - 이 가설이 사실이라고 가정하고, 단어의 목록으로부터 알파벳의 순서를 찾아내려고 한다.
- 예를 들어 다섯 개의 단어 $gg, kia, lotte, \mathrm{lg}, hanwha$ 가 사전에 순서대로 적혀 있다고 할 때, gg가 kia보다 앞에 오려면 이 언어에서는 g가 k보다 앞에 와야 함. 같은 원리로 k는 l 앞에, l은 h앞에 와야 한다는 사실을 알 수 있음. $lotte$가 lg보다 앞에 오려면 o가 g보다 앞에 와야 한다는 사실도 알 수 있음. 이를 종합하면 다섯 개의 알파벳 o, g, k, l, h 의 상대적 순서를 알게 됨.
- 사전에 포함된 단어들의 목록이 순서대로 주어질 때 이 암호사전에서 알파벳의 순서를 계산하는 프로그램을 작성하시오. 단 제시되지 않은 알파벳에 대해서는 임의 순서대로 출력해도 상관 없음.

입력

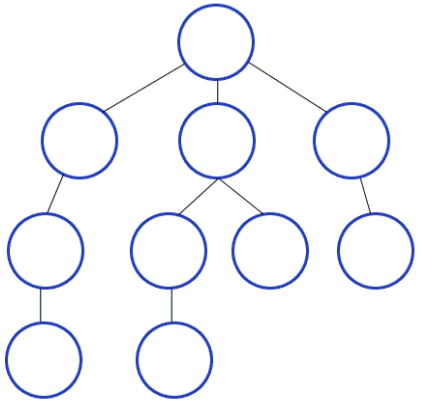
출력



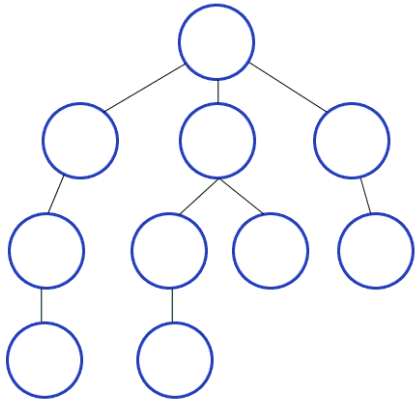
- 테스트 케이스 N
- 입력 단어 개수 K
- 단어 $w_1, w_2, \ldots w_K$

Problem 1 관련 내용:
깊이 우선 탐색 (Depth First Search)

- 트리의 순회와 같이 그래프의 모든 정점들을 특정한 순서에 따라 방문하는 알고리즘들을 그래프의 **탐색(search)** 알고리즘이라고 함.
- 그래프는 트리보다 구조가 훨씬 복잡할 수 있기 때문에 탐색 과정에서 얻어지는 정보가 아주 중요함.
- 탐색 과정에서 어떤 간선이 사용되었는지, 또 어떤 순서로 정점들이 방문되었는지를 통해 그래프의 구조를 알 수 있음.
- 탐색 알고리즘 중 가장 널리 사용되는 두 가지가 깊이 우선 탐색(DFS)과 너비 우선 탐색(BFS)임.



< 깊이 우선 탐색 >

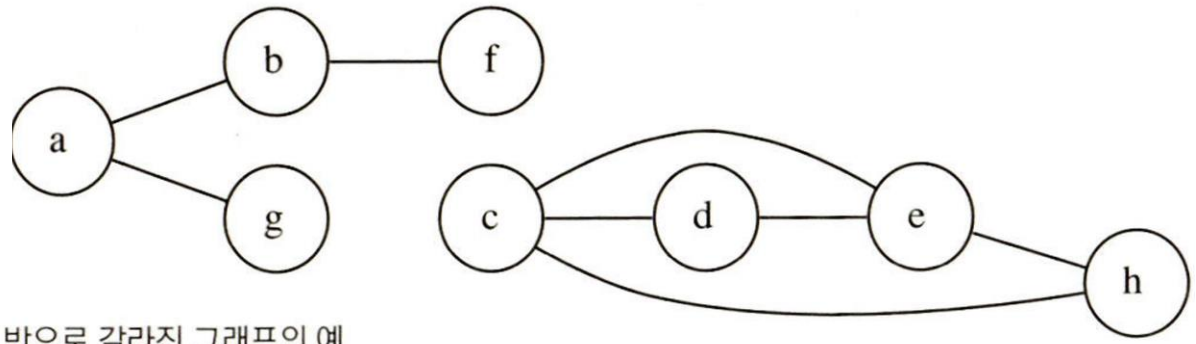


< 너비 우선 탐색 >

Problem 1 관련 내용: DFS Code

```
// 그래프의 인접 리스트 표현
vector<vector<int> > adj;
// 각 정점을 방문했는지 여부를 나타낸다.
vector<bool> visited;
// 깊이 우선 탐색을 구현한다.
void dfs(int here) {
    cout << "DFS visits " << here << endl;
    visited[here] = true;
    // 모든 인접 정점을 순회하면서
    for( int i = 0; i < adj [here]. size() ; ++i) {
        int there = adj [here] [i] ;
        // 아직 방문한 적 없다면 방문한다.
        if ( ! visited [there] )
            dfs(there) ;
    }
    // 더이상 방문할 정점이 없으니,
       재귀 호출을 종료하고 이전 정점으로 돌아간다.
}
// 모든 정점올 방문한다.
void dfsAll() {
    // visited를 모두 false로 초기화한다.
    visited = vector<bool>(adj.size( ) , false) ;
    // 모든 정점을 순회하면서 , 아직 방문한 적 없으면 방문한다.
    for(int i = 0; i < adj.size(); ++i)
        if( !visited [i] )
            dfs(i) ;
}
```

- 현재 정점과 인접한 간선들을 하나씩 검사하다가, 아직 방문하지 않은 정점으로 향하는 간선이 있다면 그 간선을 무조건 따라감. 이 과정에서 더 이상 갈 곳이 없는 막힌 정점에 도달하면 포기하고, 마지막에 따라왔던 간선을 따라 뒤로 돌아감.
- $dfsAll$: 그래프에서는 모든 정점들이 간선을 통해 연결되어 있다는 보장이 없기 때문에, dfs만으로는 모든 정점을 순서대로 발견한다는 목적에 부합하지 않음.



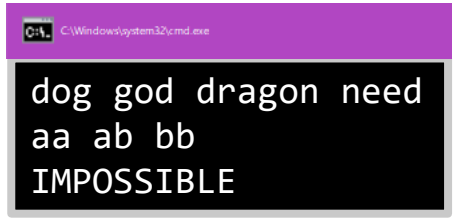
반으로 갈라진 그래프의 예

Problem 2: 영단어 끝말잇기

- 끝말잇기는 참가자들이 원을 그리고 앉은 뒤, 시계 방향으로 돌아가면서 단어를 말하는 게임임.
- 각 사람이 말하는 단어의 첫 글자는 이 전 사람이 말한 단어의 마지막 글자와 같아야 함.
- 이 프로그램에서는 일반적인 끝말잇기와 달리 사용할 수 있는 단어의 종류가 게임 시작전에 미리 정해져 있으며, 한 단어를 두 번 사용할 수 없음.
- 사용할 수 있는 단어들의 목록이 주어질 때, **단어들을 전부 사용하고 게임이 끝날 수 있는지, 그럴 수 있다면 어떤 순서로 단어를 사용해야 하는지를 계산하는 프로그램을 작성하시오.**

입력

출력



- 테스트 케이스 N
- 입력 단어 개수 K
- 단어 $w_1, w_2, \ldots w_K$

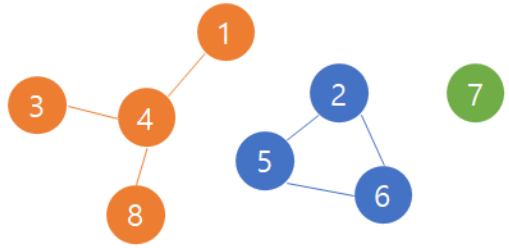
Problem 2 관련 내용: 오일러 서킷 (Eulerian circuit)

- 그래프의 모든 간선을 정확히 한 번씩 지나서 시작점으로 돌아오는 경로를 찾는 문제.
- 이와 같은 경로를 **오일러 서킷 (Eulerian circuit)**이라고 함.
- Undirected graph의 경우
  - 모든 정점이 짝수점이면서, 간선들이 하나의 컴포넌트에 포함된 그래프가 주어질 때는 항상 오일러 서킷을 찾아내는 알고리즘을 만들 수 있음.
- Directed graph의 경우
  - 모든 정점이 짝수점이면서, 간선들이 하나의 컴포넌트에 포함된 그래프가 주어질 때 는 항상 오일러 서킷을 찾아내는 알고리즘을 만들 수 있음.

    \+ 정점에 들어오는 간선의 수와 나가는 간선의 수가 같아야 함.

* 짝수점 : 짝수 단위의 간선을 가지는 정점

```
// 그래프의 안집 행렬 표현 . adj[i][j]=i와 j사이의 간선의 수
vector<vector<int>> adj;
// 무향 그래프의 인접 행렬 adj가 주어질 때 오일러 서킷을 계산한다.
// 결과로 얻어지는 circuit을 뒤집으면 오일러 서킷이 된다.
void getEulerCircuit(int here, vector<int>& circuit ) {
    for(int there = 0; there < adj.size( ); ++there)
        while(adj[here][there] > 0) {
            adj[here][there]--; // 양쪽 간선을 모두 지운다
            adj[there]here]--;
            getEulerCircuit(there, circuit) ;
        }
    circuit.push_back( here) ;
}
```



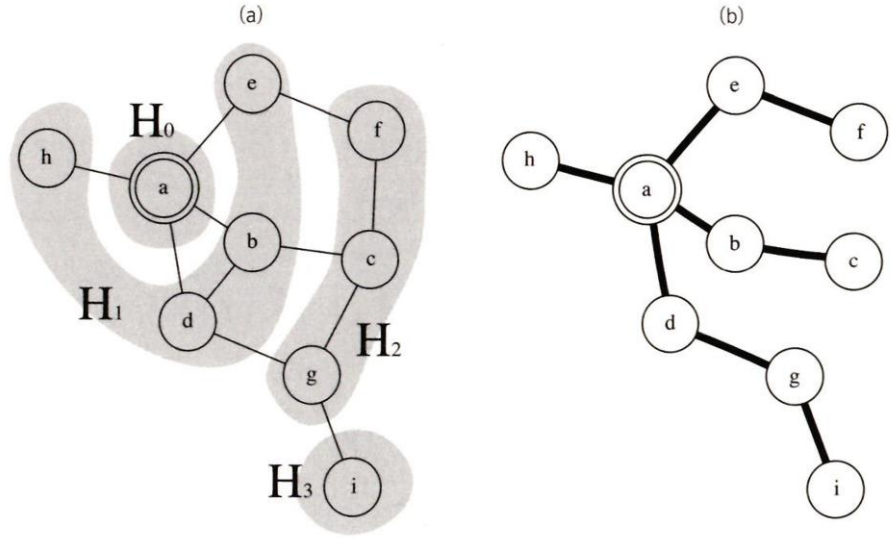
< 3개의 컴포넌트를 가진 그래프 >

Problem 2 관련 내용: 오일러 트레일 (Eulerian trail)

- 오일러 서킷처럼 모든 간선을 정확히 한번 지나지만 시작점과 끝점이 다른 경로를 찾는 문제.
- 이와 같은 경로를 오일러 트레일 (Eulerian trail)이라고 함.
- Undirected graph의 경우
  - 시작점과 끝점을 제외한 모든 점은 짝수점이고 시작점과 끝점은 홀수점이면 오일러 트레일을 찾을 수 있음.
- Directed graph의 경우
  - 시작점과 끝점을 제외한 모든 정점이 짝수점이면서, 간선들이 하나의 컴포넌트에 포함된 그래프가 주어질 때 는 항상 오일러 트레일을 찾아내 는 알고리즘을 만들 수 있음.
  - + 정점에 들어오는 간선의 수와 나가는 간선의 수가 같아야 함.

너비 우선 탐색 (Breadth First Search)

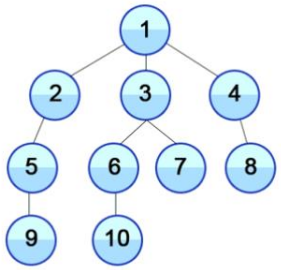
- 너비 우선 탐색은 깊이 우선 탐색과 함께 그래프 탐색 방식의 두 축을 이룸.
- 동작 과정이 그렇게 직관적이지 않은 깊이 우선 탐색에 비해, 너비 우선 탐색의 동작 과정은 아주 이해하기 쉬움. 너비 우선 탐색은 시작점에서 가까운 정점부터 순서대로 방문하는 탐색 알고리즘이기 때문임.
- (a): a를 탐색의 시작점이라고 하면, a 에서부터 최소 몇 개의 간선을 지나야 도달할 수 있는가를 기준으로 그래프의 정점들을 나눌 수 있음.
  - 간선을 i번 지나야 도착할 수 있는 정점의 집합을 H_i, 라고 부를 때, 너비 우선 탐색은 H_0 에 속한 a를 가장 먼저 방문하고, 그 후 H_1, H_2 그리고 H_3 에 속한 정점들을 순서대로 방문함.
  - 각 정점과 시작점 사이에 경로가 두 개 이상인 경우, 그중 <u>최단 경로</u>가 방문 순서를 결정하게 됨. 정점 b나 d는 a와 길이 1 인 경로로도 연결되어 있고, 2 인 경로로도 연결되어 있지만 둘 다 H_1 에 속함.
- (b): 너비 우선 탐색에서 새 정점을 발견하는 데 사용했던 간선들만을 모은 트리를 너비 우선 탐색 스패닝 트리 (BFS Spanning Tree)라고 부름.



그래프의 너비 우선 탐색

BFS Code

```
//그래프의 인접 리스트 표현
vector<vector<int> > adj;
//start에서 시작해 그래프를 너비 우선 탐색하고 각 정점의 방문 순서를 반환한다.
vector<int> bfs(int start) {
    //각 정점의 방문 여부
    vector<bool> discovered(adj.size(), false) ;
    //방문할 정점 목록을 유지하는 큐
    queue<int> q;
    //정점의 방문 순서
    vector<int> order;
    discovered[start] = true;
    q.push(start);
    while(!q.empty()) {
        int here = q.front();
        q.pop() ;
        //here를 방문한다.
        order.push_back(here) ;
        //모든 인접한 정점을 검사한다.
        for(int i=0; i<adj[here].size(); ++i) {
            int there = adj[here][i];
            // 처음 보는 정점이면 방문 목록에 집어넣는다.
            if( !discovered[there] ) {
                q.push(there) ;
                discovered[there] = true;
            }
        }
    }
    return order;
}
```

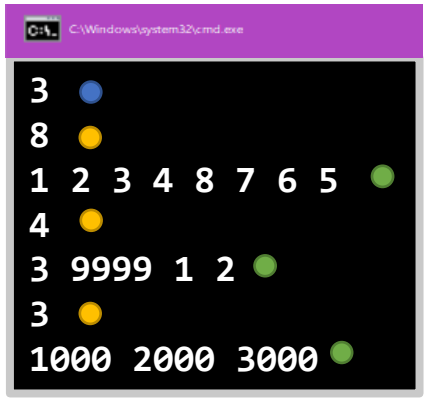


- 깊이 우선 탐색과는 달리 너비 우선 탐색에서는 발견과 방문이 같지 않음.
- 따라서 모든정점은다음과 같은 세 개의 상태를 순서대로거쳐 가게 됨.
    1. 아직 발견되지 않은 상태
    2. 발견되었지만 아직 방문되지는 않은 상태 (이 상태에 있는 정점들의 목록은 큐에 저장됨)
    3. 방문된상태
- 너비 우선 탐색은 대개 그래프에서의 최단 경로 문제를 푸는 딱 하나의 용도로 사용됨.
- 최단 경로 문제는 두 정점을 연결하는 경로 중 가장 길이가 짧은 경로를 찾는 문제로, 그래프 이론의 가장 고전적인 문제 중 하나임.
- 너비 우선탐색 알고리즘을 간단하게 변경해 모든 정점에 대해 시작점으로부터의 거리 distance[ ] 를 계산하도록 할 수 있음.

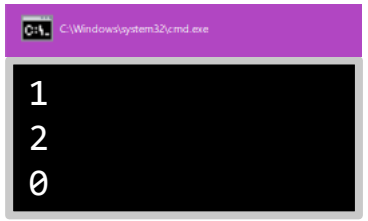
Problem 3: Sorting Game

- 정수 배열이 주어질 때 연속된 부분 구간의 순서를 뒤집는 것을 뒤집기 연산이라고 부르자.
- 예를 들어 {3, 4, 1, 2}에서 부분 구간 {4, 1, 2}를 뒤집으면 {3, 2, 1, 4}가 됨
- **<u>중복이 없는</u>** 정수 배열을 뒤집기 연산을 이용하여 오름차순으로 정렬을 하는데 필요한 최소한의 뒤집기 연산 수를 계산하는 프로그램을 작성하시오.
- 테스트 케이스 : C ≤ 1000
- 입력 배열의 길이 n : 1 ≤ n ≤ 8

입력



출력



- 테스트 케이스 C
- 입력 배열의 길이 n
- 입력 배열

과제

- 앞에서 다루지 않은 문제 중 DFS와 BFS로 풀 수 있는 문제를 각각 한 가지 씩 예를 들어 설명하시오. 관련된 그래프 구조도 그리시오.