문제해결프로그래밍실습
(CSE4152)

12 주차

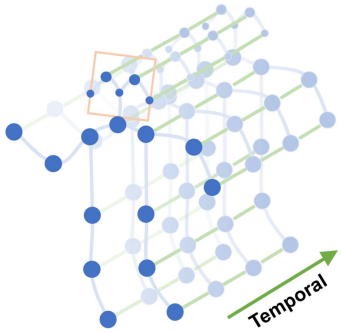
Graph 기반 문제 풀이
DFS, BFS

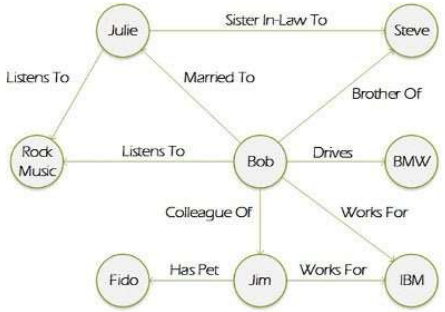
Contents

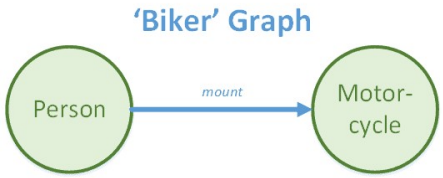
- 그래프의 표현과 정의
- 그래프의 종류와 사용 예시
- 그래프에서의 깊이 우선 탐색
- 그래프에서의 너비 우선 탐색

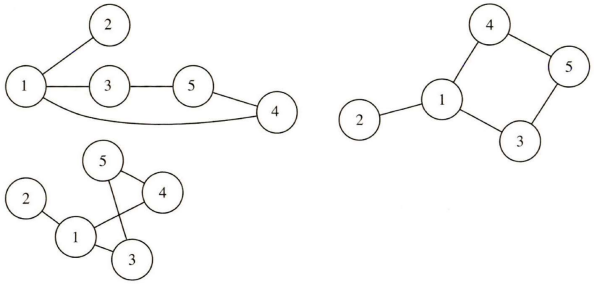
Graph의 표현과 정의

- 그래프 $G(V, E)$는 어떤 자료나 개념을 표현하는 정점 (vertex)들의 집합 V와 이 들을 연결하는 간선(edge)들의 집합 E로 구성된 자료 구조.
- 그래프는 정점들과 간선들로 정의되며, 정점의 위치 정보나 간선의 순서 등은 그래프의 정의에 포함되지 않음.
  - 따라서 오른쪽 아래의 번호 1~5의 구성을 가진 세개의 그림은 모두 같은 그래프를 표현하고 있음.
- 그래프는 현실 세계의 사물이나 추상적인 개념 간의 연결 관계를 표현함.
  - 여러 도시들을 연결하는 도로망
  - 사람들 간의 지인 관계
  - 웹사이트 간의 링크관계
- 부모 자식 관계에 관한 제약이 없기 때문에 그래프는 트리보다 훨씬 다양한 구조를 표현할 수 있고, 현실 세계의 수많은 문제들을 푸는 데 유용하게 사용됨.
  - 트리 – 방향성이 없고, 정점 간에 경로가 유일한 그래프

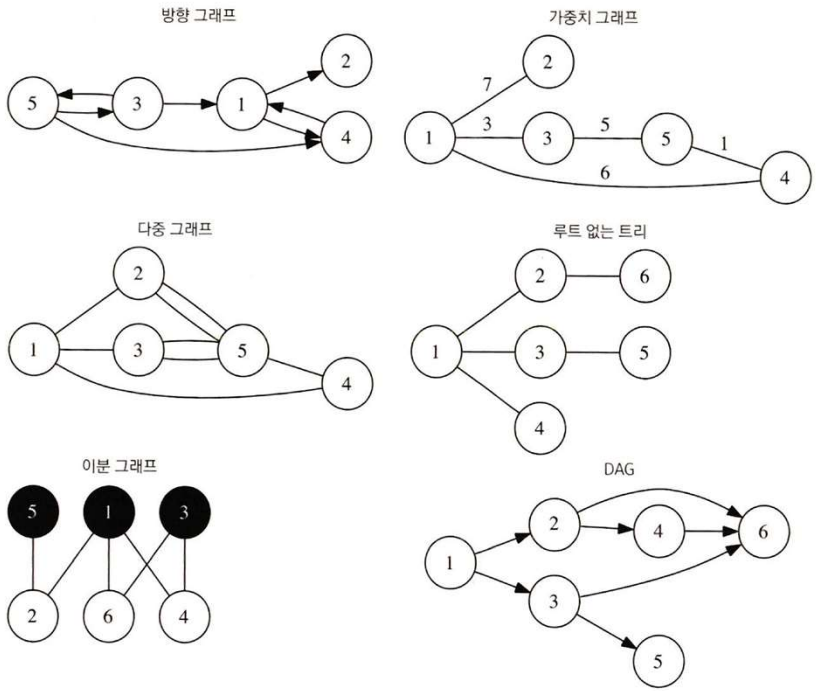








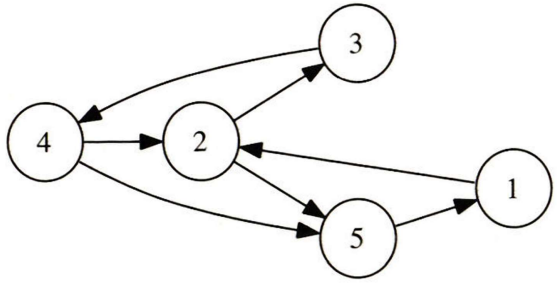
Graph의 종류



- 방향성 Directed
  - 무향 그래프 (undirected graph)
  - 방향 그래프 (directed graph)
    - 사람들간의 짝사랑 관계
    - 도로의 일방통행
- 가중치 그래프 (weighted graph) | 각 edge에 가중치(weight) 부여 가능
  - 두 도시 사이의 거리
  - 두 사람 사이의 호감도
  - 두 물건 사이의 교환 비율
- 형태
  - 단순 그래프 (simple graph) | 두 정점 사이에 최대 한 개의 간선만 있는 경우.
  - 다중 그래프 (multi graph) | 두 정점 사이에 두 개 이상의 간선이 있는 경우.
  - 이분 그래프 (bipartite graph) | 두 개의 그룹으로 나뉘어져 있고, 서로 다른 그룹의 정점 간에만 간선이 있는 경우.
  - 루트 없는 트리 (unrooted tree) | 루트가 지정되지 않은 트리.
- 사이클 없는 방향 그래프 (directed acyclic graph, DAG)
  - 한 점에서 출발해 자기 자신으로 돌아오는 경로(사이클) 가 존재하지 않는 경우. DAG는 여러 작업들 간의 상호 의존 관계 등을 그래프로 표현할 때 사용됨.

Graph의 경로 (path)

- 그래프의 경로 (path)란 끝과 끝이 서로 연결된 간선들을 순서대로 나열한 것.
- 오른쪽 그림에서 간선들 $(1,2),(2,3),(3,4),(4,5)$는 한 개의 경로를 이룸.
  - 정점의 번호만을 써서 1-2-3-4-5로 표기할 수 있음.
  - 방향 그래프이기 때문에, 1-2-4-5는 경로라고 할 수 없음.
- 경로 중 한 정점을 최대 한번만 지나는 경로를 **단순 경로 (simple path)** 라고 함.
  - 경로 2-3-4-2-5는 2번 정점을 두 번 지나기 때문에 단순 경로가 아님.
  - 현대 그래프 이론에서는 보통 "경로"를 단순 경로를 지칭하는데 사용됨.



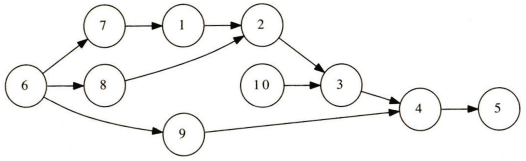
Graph의 사용 예

- 철도망의 안전 분석
    - 어떤 철도망에서 한 역이 폐쇄되어 열차가 지나지 못할경우 철도망 전체가 두 개 이상으로 쪼개질 가능성이 있는지 , 만약 있다면 어느역이 그런 위험성을 갖고 있는지 분석.
- 소셜 네트워크 분석
    - 내가 한 다리 건너 알고 있는 사람은 몇 명이나 되는지, 몇 다리나 건너가야 바이든과 내가 아는 사이인지.
- 한붓 그리기 | 오일러 경로 (Eulerian path)
    - 깊이 우선 탐색(DFS)으로 해결 가능 함.

암시적 그래프 구조 Implicit Graph Structure

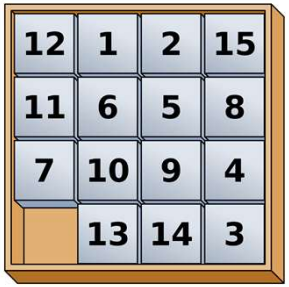
- 할 일 목록 정리 | 위상 정렬 (topology sort), 의존성 그래프

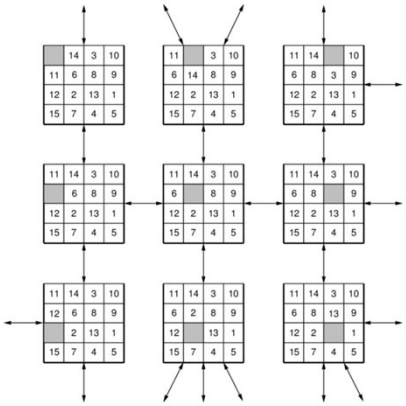
1. 김치를 다진 마늘과 볶는다.
2. 볶은 김치에 돼지고기, 고춧가루, 다진 마늘을 넣고 더 볶는다.
3. 멸치 육수를 붓고 끓인다.
4. 파와 양파를 넣고 좀더 끓인다.
5. 맛있게 먹는다.
6. 냉장고에서 재료들을 꺼낸다.
7. 김치를 썬다.
8. 돼지고기를 썬다.
9. 파와 양파를 썬다.
10. 멸치 육수를 낸다.



DAG (directed acyclic graph)

- 15-퍼즐 | DFS, BFS





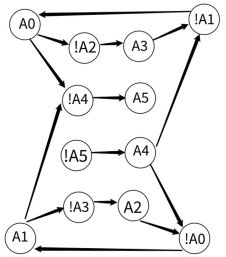
- 회의실 배정 | 2-SAT (Satisfiability) problem

start time end time (회의명)

2 5(A0) 6 9(A1)
1 3(A2) 8 10(A3)
4 7(A4) 11 12(A5)

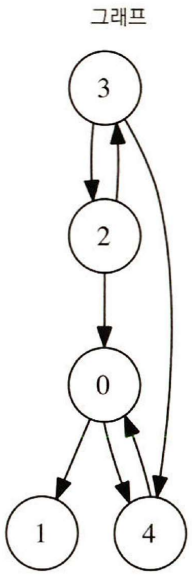
A가 진행되면, B는 진행되서는 안 되고 A => !B

B가 진행되면, A는 진행되서는 안 됩니다. B => !A



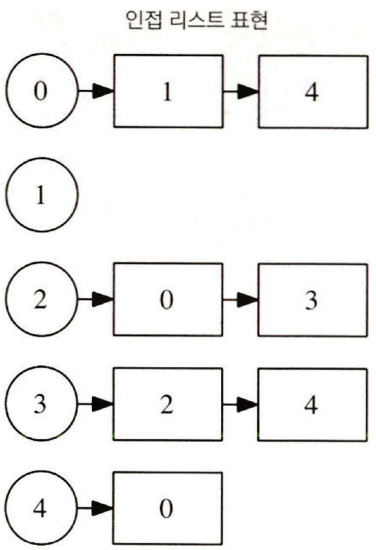
Graph의 표현 방법

- 인접 행렬 (adjacency matrix) `vector<vector<bool>> adjacent;`
- 인접 리스트 (adjacency list) `vector<list<int>> adjacent;`



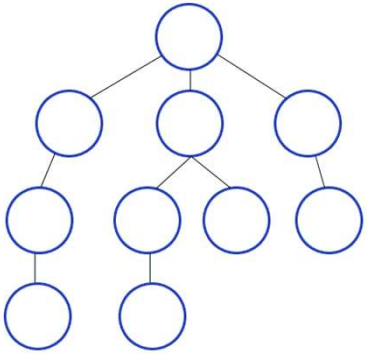
인접 행렬 표현

	0	1	2	3	4
0	0	1	0	0	1
1	0	0	0	0	0
2	1	0	0	1	0
3	0	0	1	0	1
4	1	0	0	0	0

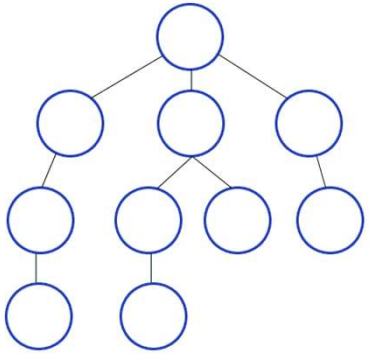


깊이 우선 탐색 (Depth First Search)

- 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 >

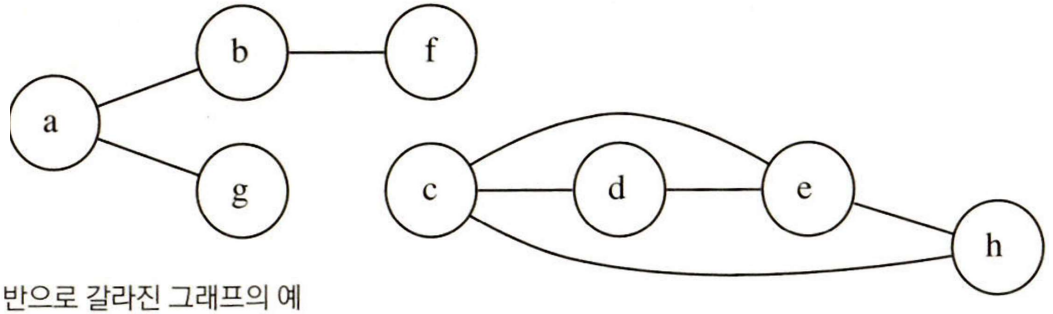


< 너비 우선 탐색 >

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

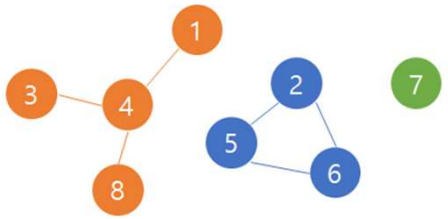
오일러 서킷 (Eulerian circuit)

- 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 인접 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 >

오일러 트레일 (Eulerian trail)

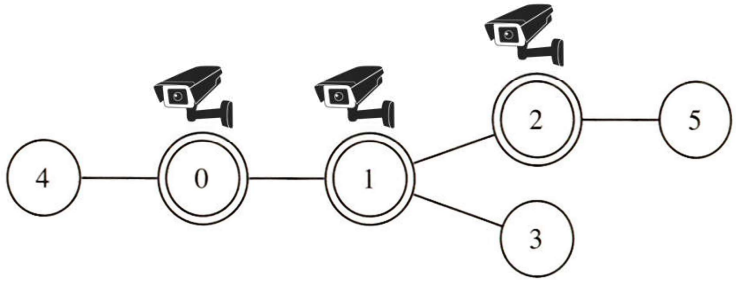
- 오일러 서킷처럼 모든 간선을 정확히 한번 지나지만 시작점과 끝점이 다른 경로를 찾는 문제.
- 이와 같은 경로를 오일러 트레일 (Eulerian trail)이라고 함.
- Undirected graph의 경우
  - 시작점과 끝점을 제외한 모든 점은 짝수점이고 시작점과 끝점은 홀수점이면 오일러 트레일을 찾을 수 있음.
- Directed graph의 경우
  - 시작점과 끝점을 제외한 모든 정점이 짝수점이면서, 간선들이 하나의 컴포넌트에 포함된 그래프가 주어질 때 는 항상 오일러 트레일을 찾아내 는 알고리즘을 만들 수 있음.
 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

지배집합

- 각 정점이 자기 자신과 모든 인접한 정점들을 지배한다고 할 때, 그래프의 모든 정점을 지배하는 정점의 부분집합을 그래프의 지배 집합(dominating set)이라고 부름.
- 코딩 과제 9-1은 아래 조건에 따라 루트 없는 트리로 볼 수 있음.
  - 정확히 V-1 개의 간선이 있음.
  - 사이클이 존재하지 않음.
  - 두 정점 사이를 연결하는 단순 경로가 정확히 하나 있음.

- 트리의 지배 집합 찾기
  - 트리의 최소 지배 집합을 찾는 가장 간단한 방법은 트리의 맨 아래에서부터 시작해서 위로 올라오는 것임. 트리에서의 각 노드의 선택 여부는 아래와 같은 알고리즘으로 결정할 수 있음.

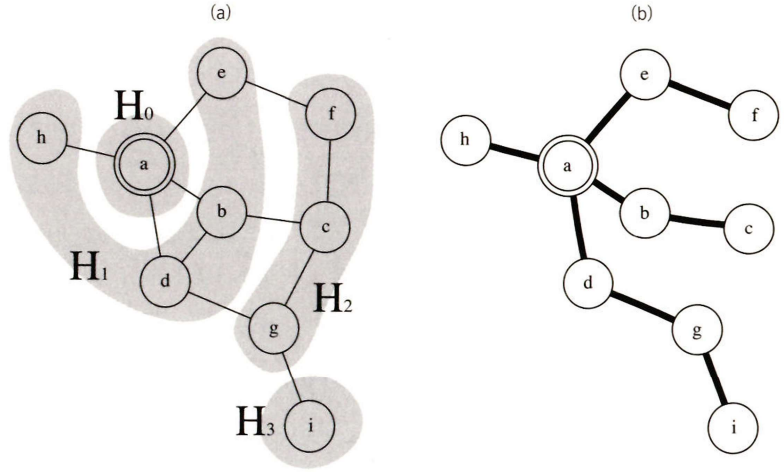
1. 잎(leaf) 노드는 선택하지 않음.
2. 이 외의 노드에 대해, 트리의 맨 밑에서부터 올라오면서 다음과 같이 선택 여부를 결정.
   a. 자기 자손(연결된 정점) 중 아직 지배당하지 않은 노드가 하나라도 있다면 현재 노드를 선택.
   b. 이 외의 경우, 현재 노드를 선택하지 않음.



첫번째 입력의 그래프

너비 우선 탐색 (Breadth First Search)

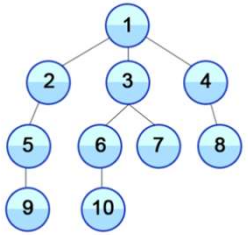
- 너비 우선 탐색은 깊이 우선 탐색과 함께 그래프 탐색 방식의 두 축을 이룸.
- 동작 과정이 그렇게 직관적이지 않은 깊이 우선 탐색에 비해, 너비 우선 탐색의 동작 과정은 아주 이해하기 쉬움. 너비 우선 탐색은 시작점에서 가까운 정점부터 순서대로 방문하는 탐색 알고리즘이기 때문임.
- (a): a를 탐색의 시작점이라고 하면, a 에서부터 최소 몇 개의 간선을 지나야 도달할 수 있는가를 기준으로 그래프의 정점들을 나눌 수 있음.
  - 간선을 i번 지나야 도착할 수 있는 정점의 집합을 H_i, 라고 부를 때, 너비 우선 탐색은 H_0 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 최단 경로가 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;
}
```

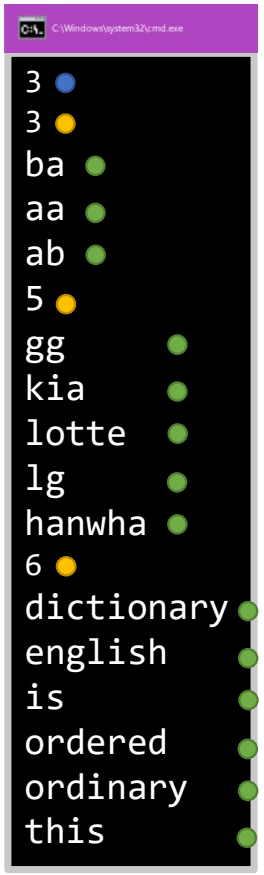


- 깊이 우선 탐색과는 달리 너비 우선 탐색에서는 발견과 방문이 같지 않음.
- 따라서 모든 정점은 다음과 같은 세 개의 상태를 순서대로 거쳐 가게 됨.
    1. 아직 발견되지 않은 상태
    2. 발견되었지만 아직 방문되지는 않은 상태 (이 상태에 있는 정점들의 목록은 큐에 저장됨)
    3. 방문된상태
- 너비 우선 탐색은 대개 그래프에서의 최단 경로 문제를 푸는 딱 하나의 용도로 사용됨.
- 최단 경로 문제는 두 정점을 연결하는 경로 중 가장 길이가 짧은 경로를 찾는 문제로, 그래프 이론의 가장 고전적인 문제 중 하나임.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

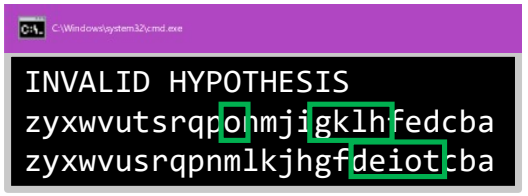
보고서 과제 9-1:
무작위로 알파벳 순서가 섞인 암호화 사전

- 범죄에 연루된 사람들의 암호화된 인명부 사전이 수사 중 발견되었다.
  - 암호화된 인명부에 포함된 단어들은 모두 영어의 **<u>소문자</u>** 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. <u>단 제시되지 않은 알파벳에 대해서는 임의 순서대로 출력해도 상관 없음.</u>

입력



출력



테스트 케이스 N

입력 단어 개수 K

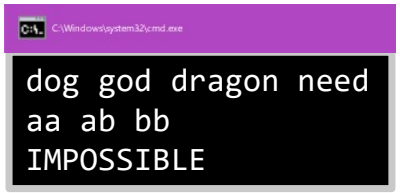
단어 $w_1, w_2, \ldots w_K$

보고서 과제 9-2: 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

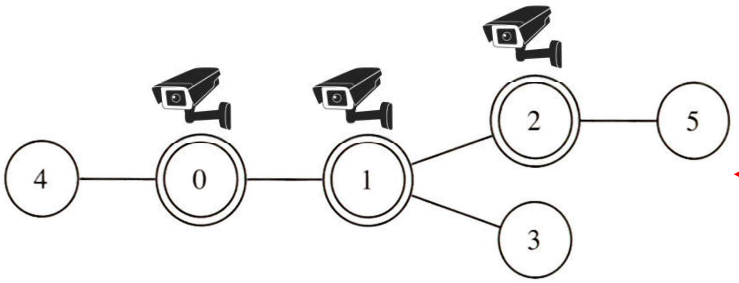
출력



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

코딩 과제 9-1: 서버보안 CCTV 설치

- 서버실은 여러 개의 방과 이들을 연결하는 복도로 구성되어 있다.
- 한 방에 CCTV를 설치하면 해당 방과 함께 해당 방과 **복도(간선)로 직접 연결된 방**들을 모두 감시할 수 있다.
- 서버실은 한 번 지나간 방에 다시 가기 위해서는 이전에 지나왔던 복도를 반드시 한 번 이상 지나야 하는 구조로 설계되어 있으며, 모든 방은 서로 연결되어 있지 않을수도 있다.
- 모든 방을 감시하기 위해 필요한 감시 카메라의 최소 개수를 구하라.



첫번째 입력의 그래프

입력

```
3
6 5
0 1
1 2
1 3
2 5
0 4
4 2
0 1
2 3
1000 1
0 1
```

출력

```
3
2
999
```

- 테스트 케이스 N
- 방, 복도 개수 R, H
- 방번호-방번호 $r_i\ r_j$

코딩 과제 9-2: 선물 나눠주기

- 내무반에 있는 n명의 해병들에게 선물세트를 나눠 주기로 함.
- 모든 해병들에게 같은 수의 선물세트를 주려고 했지만, 그 중 m $(0 \leq m < n)$명의 병장 해병들은 모두가 평등하게 같은 개수의 선물세트를 받는걸 거부함.
- 따라서 병장 해병들에게는 남들보다 선물세트를 하나씩 더 주기로 함.
- 알 수 없는 군사적인 이유로 인해, 해병들에게 나눠 주는 선물세트의 총 수 c 는 십진수 (decimal)로 썼을 때 제시된 숫자 (digit)만으로 구성되어야 함.
- 예산이 빠듯한 관계로 선물세트를 가능한 한 적게 사려고 할 때, 위와 같은 조건을 만족하는 최소의 c 를 계산하는 프로그램을 작성하시오. 단 모든 해병이 선물세트를 하나씩은 받아야 함.

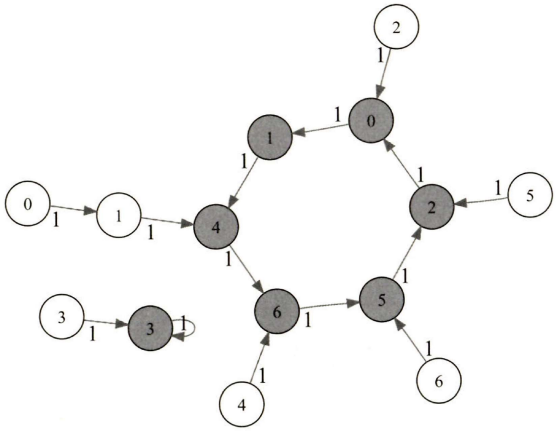
입력

```
5
1 7 0
1 10 1
0 7 3
345 9997 3333
35 9 8
```

출력

```
111111
11
IMPOSSIBLE
35355353545
35
```

- 테스트 케이스 N
- 숫자목록 d, 총인원 n, 병장수 m



문제의 첫 번째 예제 입력 그래프. 각 정점은 7로 나눈 나머지를 말함.

과제

- 보고서 과제 9 :
  - 9-1 : BFS나 DFS를 활용한 풀이를 설명
  - 9-2 : 오일러 서킷을 활용한 풀이를 설명

- 코딩 과제 9 :
  - 9-1, 9-2 슬라이드 및 Elice 참고, C++ 사용