2022-2 HI-ARC 중급스터디

3주차. 그래프 탐색

이지은 (leeju1013)

목차

* 스택, 큐 복습

스택(stack) 복습

- What?
  - 한쪽 끝에서만 원소를 넣거나 뺄 수 있는 자료구조

- How?
  - #include <stack>
  - push(x) : 원소 추가
  - pop() : 원소 제거
  - top() : 제일 상단의 원소 확인
  - empty() : 비어있는지 확인
  - size() : 원소 개수 확인

10828번

스택

성공

실버 IV

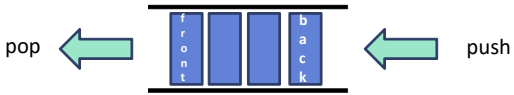
- push X: 정수 X를 스택에 넣는 연산이다.
- pop: 스택에서 가장 위에 있는 정수를 빼고, 그 수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.
- size: 스택에 들어있는 정수의 개수를 출력한다.
- empty: 스택이 비어있으면 1, 아니면 0을 출력한다.
- top: 스택의 가장 위에 있는 정수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.

```
include <iostream>
include <stack>
sing namespace std;

nt main() {
    ios_base::sync_with_stdio(false);

    stack<int> st;
    int n;
    string s;

    cin>> n;
    while(n--){
        cin >> s;
        if(s=="push"){
            int tmp; cin >> tmp;
            st.push(tmp);
        }
        else if(s=="pop"){
            if(st.empty()) cout <<"-1\n";
            else{
                cout << st.top()<<'\n';
                st.pop();
            }
        }
        else if(s=="size"){
            cout<< st.size()<<'\n';
        }
        else if(s=="empty"){
            cout << st.empty() <<'\n';
            //스택이 비어있으면 1, 아니면 0을 반환함
        }
        else if(s=="top"){
            if(st.empty()) cout<<"-1\n";
            else cout << st.top()<<'\n';
        }
    }
    return 0;
}
```

큐(queue) 복습

- What?
  - 한쪽 끝에서 원소를 넣고 반대쪽 끝에서 원소를 뺄 수 있는 자료구조

- How?
  - #include <queue>
  - push(x) : 맨 뒤에 원소 추가
  - pop() : 맨 앞의 원소 제거
  - front() : 맨 앞의 원소 확인
  - back() : 맨 뒤의 원소 확인
  - empty() : 비어있는지 확인
  - size() : 원소 개수 확인



10845번

큐

성공

실버 IV

- push X: 정수 X를 큐에 넣는 연산이다.
- pop: 큐에서 가장 앞에 있는 정수를 빼고, 그 수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
- size: 큐에 들어있는 정수의 개수를 출력한다.
- empty: 큐가 비어있으면 1, 아니면 0을 출력한다.
- front: 큐의 가장 앞에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
- back: 큐의 가장 뒤에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.

```cpp
#include <iostream>
#include <queue>
using namespace std;
int main() {
    ios_base::sync_with_stdio(false);

    queue<int> q;
    int n; cin >> n;
    string s;

    for(int i=0; i<n; i++){
        cin >> s ;
        if(s=="push"){
            int tmp; cin >> tmp;
            q.push(tmp);
        }
        else if(s=="front"){
            if(q.empty()) cout << "-1\n";
            else  cout << q.front() << '\n';
        }
        else if(s=="back"){
            if(q.empty()) cout << "-1\n";
            else cout << q.back()<<'\n';
        }
        else if(s=="size"){
            cout << q.size()<<'\n';
        }
        else if(s=="empty"){
            cout << q.empty() << '\n';
        }
        else if(s=="pop"){
            if(q.empty()) cout << "-1\n";
            else{
                cout << q.front() << '\n';
                q.pop();
            }
        }
    }
    return 0;
}
```

바이러스

성공

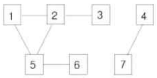
실버 III

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	98051	46574	31449	45.811%

문제

신종 바이러스인 웜 바이러스는 네트워크를 통해 전파된다. 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.

예를 들어 7대의 컴퓨터가 <그림 1>과 같이 네트워크 상에서 연결되어 있다고 하자. 1번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 2번과 5번 컴퓨터를 거쳐 3번과 6번 컴퓨터까지 전파되어 2, 3, 5, 6 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 4번과 7번 컴퓨터는 1번 컴퓨터와 네트워크상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.



< 그림 1 >

예제 입력 1 복사

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

예제 출력 1 복사

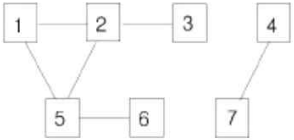
```
4
```

어느 날 1번 컴퓨터가 웜 바이러스에 걸렸다. 컴퓨터의 수와 네트워크 상에서 서로 연결되어 있는 정보가 주어질 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

1. 그래프

- What?

- **정점**(**V**ertex)과 **간선**(**E**dge)의 집합으로 이루어진 자료구조

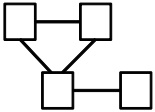


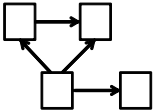
< 그림 1 >

- 정점 : 1, 2, 3, 4, 5, 6, 7
- 간선 : (1, 2), (2, 3), (1, 5), (5, 2), (5, 6), (4, 7)

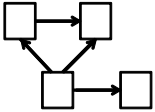
- 인접(adjacent) : 간선으로 이어진 두 정점
- 차수(degree) : 정점에 연결된 간선의 수

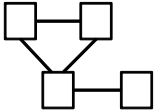
- 방향성 : 무방향 그래프 방향그래프



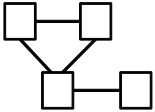


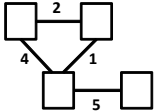
- 사이클 : 비순환 그래프 순환 그래프





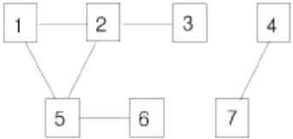
- 가중치 : 비가중 그래프 가중 그래프





1. 그래프

- How? _ 인접 행렬 int adj[8][8]



< 그림 1 >

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

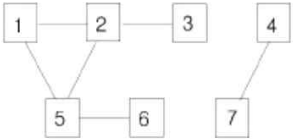
- 인접하면 1, 인접하지 않으면 0

- 장점: 간선 여부를 O(1)에 확인
- 단점: 공간 복잡도 O(| V^2|)

-> 간선 수가 많은 경우 사용

1. 그래프

- How? _ 인접 리스트 vector<int> adj[8]



< 그림 1 >

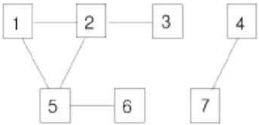
1	2	5	
2	1	3	5
3	2		
4	7		
5	1	2	6
6	5		
7	4		

- 해당 점점에서 나가는 간선 저장

- 장점: 공간 복잡도 O(|V|+|E|)
- 단점: 간선 여부를 O(degree)에 확인

-> 정점 수가 많은 경우 사용

* 대부분 문제들이 간선에 비해 정점 개수가 많기도 하고,
  특정 노드에 연결된 모든 노드를 찾는 경우가 많음
-> 공간도 적게 사용하며, 탐색시간도 빠른 인접 리스트를 주로 사용!



< 그림 1 >

예제 입력 1 복사

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

입력

첫째 줄에는 컴퓨터의 수가 주어진다. 컴퓨터의 수는 100 이하이고 각 컴퓨터에는 1번 부터 차례대로 번호가 매겨진다. 둘째 줄에는 네트워크 상에서 직접 연결되어 있는 컴퓨터 쌍의 수가 주어진다. 이어서 그 수만큼 한 줄에 한 쌍씩 네트워크 상에서 직접 연결되어 있는 컴퓨터의 번호 쌍이 주어진다.

```
int adj[8][8]={};
int v,e;
cin>> v >> e;
while(e--){
    int a,b;
    cin>> a >> b;
    adj[a][b]=1;
    adj[b][a]=1;
}
```

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

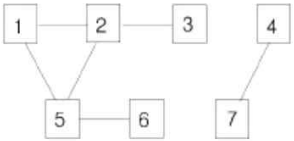
<인접 행렬>

```
vector<int> adj[8];
int v,e;
cin>> v >> e;
while(e--){
    int a,b;
    cin>> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
}
```

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

<인접 리스트>

2. 그래프 탐색



< 그림 1 >

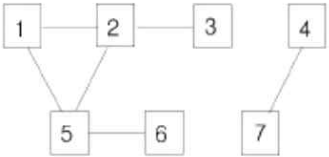
- DFS(깊이 우선 탐색)
  : 1 -> 2 -> 3 -> 5 -> 6

- BFS(너비 우선 탐색)
  : 1 -> 2 -> 5 -> 3 -> 6

2-1. DFS (Depth First Search)

- What?
  - ‘깊이’를 우선적으로 탐색하는 알고리즘

- How?
  - 스택
  - 재귀함수



- DFS(깊이 우선 탐색)
  : 1 -> 2 -> 3 -> 5 -> 6

1
5 2
5 3
5
6

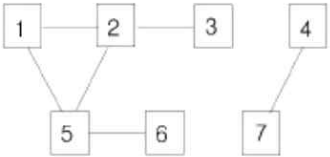
1번 정점에서 시작!, visited[1]=1, push(1)
1번 정점 방문하고 pop(),
1번 정점과 인접하고 visited[x]==0 이면 visited[x]=1, push(x)

2번 정점 방문하고 pop(),
2번 정점과 인접하고 visited[x]==0 이면 visited[x]=1, push(x)

3번 정점 방문하고 pop(),
3번 정점과 인접하고 visited[x]==0 인 정점이 없으므로 그냥 넘어감

5번 정점 방문하고 pop(),
5번 정점과 인접하고 visited[x]==0 이면 visited[x]=1, push(x)

6번 정점 방문하고 pop(),
6번 정점과 인접하고 visited[x]==0 인 정점이 없으므로 그냥 넘어감. 탐색 끝!



- DFS (깊이 우선 탐색)
  : 1 -> 2 -> 3 -> 5 -> 6

1

5 2

5 3

5

6

```cpp
vector<int> adj[8];
int visited[8]={};

void dfs(int start){
    stack<int> st;
    st.push(start);
    visited[start]=1;
    while(!st.empty()){
        int cur = st.top(); st.pop();
        for(int next : adj[cur]){
            if(!visited[next]){
                visited[next] = 1;
                st.push(next);
            }
        }
    }
}
```

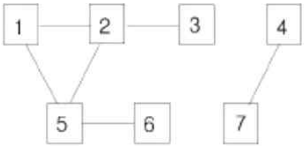
<스택>

```cpp
vector<int> adj[8];
int visited[8]={};

void dfs(int cur){
    visited[cur]=1;
    for(auto &next : adj[cur])
        if(!visited[next]) dfs(next);
}
```

<재귀함수>

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

시간복잡도: O(V+E)



입력

첫째 줄에는 컴퓨터의 수가 주어진다. 컴퓨터의 수는 100 이하이고 각 컴퓨터에는 1번 부터 차례대로 번호가 매겨진다. 둘째 줄에는 네트워크 상에서 직접 연결되어 있는 컴퓨터 쌍의 수가 주어진다. 이어서 그 수만큼 한 줄에 한 쌍씩 네트워크 상에서 직접 연결되어 있는 컴퓨터의 번호 쌍이 주어진다.

출력

1번 컴퓨터가 웜 바이러스에 걸렸을 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 첫째 줄에 출력한다.

예제 입력 1 복사

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

예제 출력 1 복사

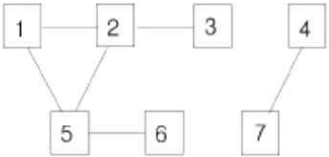
```
4
```

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

vector<int> adj[101];
int visited[101];
int ans;

void dfs(int cur){
    visited[cur]=1;
    ans++;
    for(auto &next:adj[cur]){
        if(!visited[next]) dfs(next);
    }
}
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int v,e; cin>> v >> e;
    while(e--){
        int a,b; cin>> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    dfs(1);
    cout<<ans-1; //1번 컴퓨터는 제외
    return 0;
}
```

2-2. BFS (Breadth First Search)

- What?
  - '너비'를 우선적으로 탐색하는 알고리즘

- How?
  - 큐



- BFS (너비 우선 탐색)
  : 1 -> 2 -> 5 -> 3 -> 6

1
2 5
5 3
3 6
6

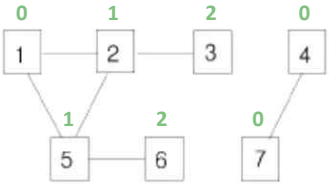
1번 정점에서 시작!, visited[1]=1, push(1)
1번 정점 방문하고 pop(),
1번 정점과 인접하고 visited[x]==0 이면 visited[x]=1, push(x)

2번 정점 방문하고 pop(),
2번 정점과 인접하고 visited[x]==0 이면 visited[x]=1, push(x)

5번 정점 방문하고 pop(),
5번 정점과 인접하고 visited[x]==0 이면 visited[x]=1, push(x)

3번 정점 방문하고 pop(),
3번 정점과 인접하고 visited[x]==0 인 정점이 없으므로 그냥 넘어감

6번 정점 방문하고 pop(),
6번 정점과 인접하고 visited[x]==0 인 정점이 없으므로 그냥 넘어감. 탐색 끝!



- BFS (너비 우선 탐색)
  : 1 -> 2 -> 5 -> 3 -> 6

```
1 vector<int> adj[8];
2 int visited[8]={};
3 int dist[8]={};
4
5 void bfs(int start){
6     queue<int> q;
7     q.push(start);
8     visited[start]=1;
9     while(!q.empty()){
10        int cur = q.top(); q.pop();
11        for(int next : adj[cur]){
12            if(!visited[next]){
13                visited[next] = 1;
14                dist[next] = dist[cur]+1; //최단거리
15                q.push(next);
16            }
17        }
18    }
19 }
```

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

시간복잡도: O(V+E)

<큐>

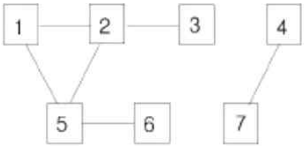
1

2 5

5 3

3 6

6



입력

첫째 줄에는 컴퓨터의 수가 주어진다. 컴퓨터의 수는 100 이하이고 각 컴퓨터에는 1번 부터 차례대로 번호가 매겨진다. 둘째 줄에는 네트워크 상에서 직접 연결되어 있는 컴퓨터 쌍의 수가 주어진다. 이어서 그 수만큼 한 줄에 한 쌍씩 네트워크 상에서 직접 연결되어 있는 컴퓨터의 번호 쌍이 주어진다.

출력

1번 컴퓨터가 웜 바이러스에 걸렸을 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 첫째 줄에 출력한다.

예제 입력 1 복사

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

예제 출력 1 복사

```
4
```

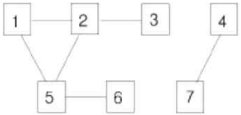
```cpp
vector<int> adj[101];
int visited[101];
int ans;

void bfs(int start){
    queue<int> q;
    q.push(start);
    visited[start]=1;
    while(!q.empty()){
        int cur = q.front(); q.pop();
        for(int next:adj[cur]){
            if(!visited[next]){
                visited[next] = 1;
                ans++;
                q.push(next);
            }
        }
    }
}
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int v,e; cin>> v >> e;
    while(e--){
        int a,b; cin>> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    bfs(1);
    cout<<ans;
    return 0;
}
```

DFS vs BFS

- 깊이 우선 탐색
- 스택 (재귀함수)
- 사이클 검출
- 1 -> 2 -> 3 -> 5 -> 6

- 너비 우선 탐색
- 큐
- 최단거리 찾기
- 1 -> 2 -> 5 -> 3 -> 6

```
vector<int> adj[8];
int visited[8]={};

void dfs(int cur){
    visited[cur]=1;
    for(auto &next : adj[cur])
        if(!visited[next]) dfs(next);
}
```




```
vector<int> adj[8];
int visited[8]={};
int dist[8]={};

void bfs(int start){
    queue<int> q;
    q.push(start);
    visited[start]=1;
    while(!q.empty()){
        int cur = q.top(); q.pop();
        for(int next : adj[cur]){
            if(!visited[next]){
                visited[next] = 1;
                dist[next] = dist[cur]+1; //최단거리
                q.push(next);
            }
        }
    }
}
```

쉬는 시간

* 스택, 큐 복습

1. 그래프
2. 그래프 탐색
   1) DFS
   2) BFS
   - 2606. 바이러스
   - 18352. 특정거리의 도시찾기
   - 1926. 그림
   - 7576. 토마토

특정 거리의 도시 찾기 성공

실버 II

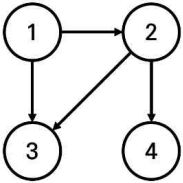
시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	256 MB	27986	8535	5426	28.934%

문제

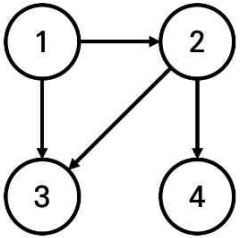
어떤 나라에는 1번부터 N번까지의 도시와 M개의 단방향 도로가 존재한다. 모든 도로의 거리는 1이다.

이 때 특정한 도시 X로부터 출발하여 도달할 수 있는 모든 도시 중에서, 최단 거리가 정확히 K인 모든 도시들의 번호를 출력하는 프로그램을 작성하시오. 또한 출발 도시 X에서 출발 도시 X로 가는 최단 거리는 항상 0이라고 가정한다.

예를 들어 N=4, K=2, X=1일 때 다음과 같이 그래프가 구성되어 있다고 가정하자.



이 때 1번 도시에서 출발하여 도달할 수 있는 도시 중에서, 최단 거리가 2인 도시는 4번 도시 뿐이다. 2번과 3번 도시의 경우, 최단 거리가 1이기 때문에 출력하지 않는다.



입력

첫째 줄에 도시의 개수 N, 도로의 개수 M, 거리 정보 K, 출발 도시의 번호 X가 주어진다. ($2 \le N \le 300,000$, $1 \le M \le 1,000,000$, $1 \le K \le 300,000$, $1 \le X \le N$) 둘째 줄부터 M개의 줄에 걸쳐서 두 개의 자연수 A, B가 공백을 기준으로 구분되어 주어진다. 이는 A번 도시에서 B번 도시로 이동하는 단방향 도로가 존재한다는 의미다. ($1 \le A, B \le N$) 단, A와 B는 서로 다른 자연수이다.

출력

X로부터 출발하여 도달할 수 있는 도시 중에서, 최단 거리가 K인 모든 도시의 번호를 한 줄에 하나씩 오름차순으로 출력한다.

이 때 도달할 수 있는 도시 중에서, 최단 거리가 K인 도시가 하나도 존재하지 않으면 -1을 출력한다.

예제 입력 1 복사

```
4 4 2 1
1 2
1 3
2 3
2 4
```

예제 출력 1 복사

```
4
```

예제 입력 2 복사

```
4 3 2 1
1 2
1 3
1 4
```

예제 출력 2 복사

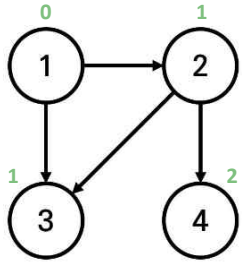
```
-1
```

예제 입력 3 복사

```
4 4 1 1
1 2
1 3
2 3
2 4
```

예제 출력 3 복사

```
2
3
```



1	2	3
2	3	4
3		
4		

1

2 3

3 4

4

idx	1	2	3	4
dist	0	1	1	2

x=1일때

```cpp
int n,m,k,x;
vector<int> adj[NMAX+1]; //1based
int visited[NMAX+1];
int dist[NMAX+1];

void bfs(int start){
    queue<int>q;
    q.push(start); //출발도시
    visited[start]=1;
    while(!q.empty()){
        int cur=q.front(); q.pop();
        for(auto next : adj[cur]){
            if(!visited[next]){
                visited[next]=1;
                dist[next] = dist[cur]+1; //최단거리 계산
                q.push(next);
            }
        }
    }
}
```

예제 입력 1 복사

```
4 4 2 1
1 2
1 3
2 3
2 4
```

예제 출력 1 복사

```
4
```

```cpp
#include <iostream>
#include <queue>
#include <vector>
#define NMAX 300000
using namespace std;

int n,m,k,x;
vector<int> adj[NMAX+1]; //1based
int visited[NMAX+1];
int dist[NMAX+1];

void bfs(int start){
    queue<int>q;
    q.push(start); //출발도시
    visited[start]=1;
    while(!q.empty()){
        int cur=q.front(); q.pop();
        for(auto next : adj[cur]){
            if(!visited[next]){
                visited[next]=1;
                dist[next] = dist[cur]+1; //최단거리 계산
                q.push(next);
            }
        }
    }
}
int main(){
    ios::sync_with_stdio(0); cin.tie(0);

    cin >>n>>m>>k>>x;
    while(m--){
        int a,b; cin>>a>>b;
        adj[a].push_back(b);//유향간선
    }

    bfs(x);

    bool exist=0;
    for(int i=1; i<=n; i++){
        if(dist[i]==k){
            cout <<i<<'\n';
            exist=1;
        }
    }
    if(!exist)
        cout <<-1;
    return 0;
}
```

그림

성공

실버 I

시간 제한	메모리 제한	제출	정답	맞
2 초	128 MB	14777	6355	44

문제

어떤 큰 도화지에 그림이 그려져 있을 때, 그 그림의 개수와, 그 그림 중 넓이가 가장 넓은 것의 넓이를 출력하여라. 단, 그림이라는 것은 1로 연결된 것을 한 그림이라고 정의하자. 가로나 세로로 연결된 것은 연결이 된 것이고 대각선으로 연결이 된 것은 떨어진 그림이다. 그림의 넓이란 그림에 포함된 1의 개수이다.

입력

첫째 줄에 도화지의 세로 크기 n(1 ≤ n ≤ 500)과 가로 크기 m(1 ≤ m ≤ 500)이 차례로 주어진다. 두 번째 줄부터 n+1 줄 까지 그림의 정보가 주어진다. (단 그림의 정보는 0과 1이 공백을 두고 주어지며, 0은 색칠이 안된 부분, 1은 색칠이 된 부분을 의미한다)

출력

첫째 줄에는 그림의 개수, 둘째 줄에는 그 중 가장 넓은 그림의 넓이를 출력하여라. 단, 그림이 하나도 없는 경우에는 가장 넓은 그림의 넓이는 0이다.

예제 입력 1 복사

```
6 5
1 1 0 1 1
0 1 1 0 0
0 0 0 0 0
1 0 1 1 1
0 0 1 1 1
0 0 1 1 1
```

예제 출력 1 복사

```
4
9
```

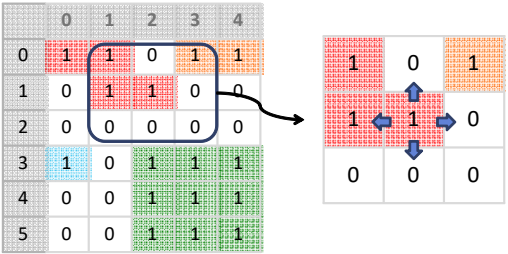
예제 입력 1 복사

```
6 5
1 1 0 1 1
0 1 1 0 0
0 0 0 0 0
1 0 1 1 1
0 0 1 1 1
0 0 1 1 1
```

예제 출력 1 복사

```
4
9
```

그림의 개수
그 중 가장 넓은 그림의 넓이




```cpp
int matrix[501][501];
int visited[501][501];
int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};
int ans;

void bfs(int x, int y){
    queue<pair<int,int>> q;
    q.push({x,y});
    visited[x][y]=1;

    while(!q.empty()){
        int cur_x=q.front().first;
        int cur_y=q.front().second;
        q.pop();

        for(int i=0; i<4; i++){
            int nx=cur_x+dx[i], ny=cur_y+dy[i];
            if(nx<0 || ny<0 || nx>=n || ny>=m) continue;
            if(matrix[nx][ny] && !visited[nx][ny]){
                visited[nx][ny]=1;
                q.push({nx,ny});
            }
        }
    }
}
```

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>

using namespace std;
int n, m, ans,cnt; //행, 열, 그림 개수, 그림 넓이
int matrix[501][501]; // 0 : 그림 없음, 1 : 그림 있음, 2 : 그림 방문했음
int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};
vector<int> space; //그림들의 넓이 보관

void bfs(int x, int y){
    queue<pair<int,int>> q;
    matrix[x][y]=2; //방문 체크
    cnt=1; //현재 그림의 넓이 초기화
    q.push({x,y});

    while(!q.empty()){
        int cur_x=q.front().first;
        int cur_y=q.front().second;
        q.pop();

        for(int i=0; i<4; i++){
            int nx=cur_x+dx[i], ny=cur_y+dy[i];
            if(nx<0 || ny<0 || nx>=n || ny>=m) continue;
            if(matrix[nx][ny]==1){ //그림 있고 아직 방문 전이면
                matrix[nx][ny] = 2; //방문 체크
                cnt++; //현재 그림 넓이 계산
                q.push({nx, ny});
            }
        }
    }
    space.push_back(cnt); //현재 그림 넓이 보관
}

int main(){
    ios_base::sync_with_stdio(false); ci

    cin>>n>>m;
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            cin>>matrix[i][j];
        }
    }

    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(matrix[i][j]==1){ //그림 있고 아직 방문 전이면
                bfs(i,j);
                ans++; // 그림의 개수
            }
        }
    }
    cout<< ans <<'\n';

    if(!ans) cout<<0; // 그림이 하나도 없는 경우에는 가장 넓은 그림의 넓이는 0
    else {
        sort(space.begin(), space.end());
        cout << space[ans - 1]; // 가장 넓은 그림의 넓이
    }
    return 0;
}
```

예제 입력 1 복사

```
6 5
1 1 0 1 1
0 1 1 0 0
0 0 0 0 0
1 0 1 1 1
0 0 1 1 1
0 0 1 1 1
```

예제 출력 1 복사

```
4
9
```

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;
int n, m, ans,cnt; //행, 열, 그림 개수, 그림 넓이
int matrix[501][501]; // 0 : 그림 없음, 1 : 그림 있음, 2 : 그림 방문
int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};
vector<int> space; //그림들의 넓이 보관

void dfs(int x, int y){
    matrix[x][y]=2; //방문 체크
    cnt++; //현재 그림 넓이 계산

    for(int i=0; i<4; i++){
        int nx=x+dx[i], ny=y+dy[i];
        if(nx<0 || ny<0 || nx>=n || ny>=m) continue;
        if(matrix[nx][ny]==1){ //그림 있고 아직 방문 전이면
            dfs(nx, ny);
        }
    }
}
int main(){
    ios_base::sync_with_stdio(false)

    cin>>n>>m;
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            cin>>matrix[i][j];
        }
    }

    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(matrix[i][j]==1){ //그림 있고 아직 방문 전이면
                cnt=0; //현재 그림의 넓이 초기화
                dfs(i,j);
                space.push_back(cnt); //현재 그림 넓이 보관
                ans++; // 그림의 개수
            }
        }
    }
    cout<< ans <<'\n';

    if(!ans) cout<<0; // 그림이 하나도 없는 경우에는 가장 넓은 그림의 넓이는 0
    else {
        sort(space.begin(), space.end());
        cout << space[ans - 1]; // 가장 넓은 그림의 넓이
    }
    return 0;
}
```

예제 입력 1 복사

```
6 5
1 1 0 1 1
0 1 1 0 0
0 0 0 0 0
1 0 1 1 1
0 0 1 1 1
0 0 1 1 1
```

예제 출력 1 복사

```
4
9
```

토마토

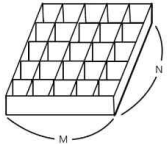
성공

골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	134627	50056	31596	35.155%

문제

철수의 토마토 농장에서는 토마토를 보관하는 큰 창고를 가지고 있다. 토마토는 아래의 그림과 같이 격자 모양 상자의 칸에 하나씩 넣어서 창고에 보관한다.



창고에 보관되는 토마토들 중에는 잘 익은 것도 있지만, 아직 익지 않은 토마토들도 있을 수 있다. 보관 후 하루가 지나면, 익은 토마토들의 인접한 곳에 있는 익지 않은 토마토들은 익은 토마토의 영향을 받아 익게 된다. 하나의 토마토의 인접한 곳은 왼쪽, 오른쪽, 앞, 뒤 네 방향에 있는 토마토를 의미한다. 대각선 방향에 있는 토마토들에게는 영향을 주지 못하며, 토마토가 혼자 저절로 익는 경우는 없다고 가정한다. 철수는 창고에 보관된 토마토들이 며칠이 지나면 다 익게 되는지, 그 최소 일수를 알고 싶어 한다.

토마토를 창고에 보관하는 격자모양의 상자들의 크기와 익은 토마토들과 익지 않은 토마토들의 정보가 주어졌을 때, 며칠이 지나면 토마토들이 모두 익는지, 그 최소 일수를 구하는 프로그램을 작성하라. 단, 상자의 일부 칸에는 토마토가 들어있지 않을 수도 있다.

입력

첫 줄에는 상자의 크기를 나타내는 두 정수 M,N이 주어진다. M은 상자의 가로 칸의 수, N은 상자의 세로 칸의 수를 나타낸다. 단, 2 ≤ M,N ≤ 1,000 이다. 둘째 줄부터는 하나의 상자에 저장된 토마토들의 정보가 주어진다. 즉, 둘째 줄부터 N개의 줄에는 상자에 담긴 토마토의 정보가 주어진다. 하나의 줄에는 상자 가로줄에 들어있는 토마토의 상태가 M개의 정수로 주어진다. 정수 1은 익은 토마토, 정수 0은 익지 않은 토마토, 정수 -1은 토마토가 들어있지 않은 칸을 나타낸다.

토마토가 하나 이상 있는 경우만 입력으로 주어진다.

출력

여러분은 토마토가 모두 익을 때까지의 최소 날짜를 출력해야 한다. 만약, 저장될 때부터 모든 토마토가 익어있는 상태이면 0을 출력해야 하고, 토마토가 모두 익지는 못하는 상황이면 -1을 출력해야 한다.

예제 입력 1 복사

```
6 4
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
```

예제 출력 1 복사

```
8
```

예제 입력 2 복사

```
6 4
0 -1 0 0 0 0
-1 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
```

예제 출력 2 복사

```
-1
```

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1

BFS

9	8	7	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2
6	5	4	3	2	1

모두 익는 데에 9-1 =8일 걸림
8 출력

0	-1	0	0	0	0
-1	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1

BFS

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

모두 익지 못하면
(=BFS가 끝난 후 0인 토마토가 있으면)
-1출력

1	-1	0	0	0	0
0	-1	0	0	0	0
0	0	0	0	-1	0
0	0	0	0	-1	1

BFS

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

모두 익는 데에 7-1 = 6일 걸림
6 출력

```cpp
#include <iostream>
#include <queue>
using namespace std;
int m,n,day; // 가로, 세로, 최소날짜
int tomato[1001][1001];
    // 1:익은 토마토, 0:익지 않은 토마토, -1: 토마토가 들어있지 않은 칸
int dx[4]={-1,0,1,0};
int dy[4]={0,1,0,-1};

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

    cin>>m>>n; //가로, 세로

    // 토마토 입력받기
    queue<pair<int,int>> q;
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            cin >> tomato[i][j];
            if(tomato[i][j]==1){ //토마토 익었으면
                q.push({i,j}); //큐에 넣어둠
            }
        }
    }
    while(!q.empty()){ //큐가 빌때까지 뽑자
        int cur_x = q.front().first;
        int cur_y = q.front().second;
        q.pop();

        for(int i=0; i<4; i++){
            int nx=cur_x+dx[i], ny=cur_y+dy[i];
            if(nx<0 || ny<0 || nx>=n || ny>=m) continue;
            if(tomato[nx][ny]==0){ //주변에 안익은 토마토 있으면
                q.push({nx,ny}); //큐에넣음
                tomato[nx][ny]=tomato[cur_x][cur_y]+1; //날짜 하루증가
            }
        }
    }

    //토마토 다 익었는지, 며칠 걸렸는지 확인
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            day = max(day, tomato[i][j]);
            if(tomato[i][j]==0) { //안익은 토마토 있으면 즉시종료
                cout << -1;
                return 0;
            }
        }
    }

    cout << day-1;
    //1부터 시작해서 하루씩 증가하도록 tomato[x][y]에 넣어줬는데
    //처음에 1은 0일차 이므로 day-1을 출력해줘야함
    return 0;
}
```

감사합니다

- 필수 문제
  - 2606. 바이러스
  - 18352. 특정거리의도시찾기
  - 1926. 그림
  - 7576. 토마토
- 연습 문제
  - 10828. 스택
  - 10845. 큐
  - 2178. 미로 탐색
  - 7562. 나이트의 이동
  - 1012 유기농배추
  - 4963 섬의 개수
  - 2583 영역구하기
- 10월 12일(수요일) 저녁 6시 T702