#중급반 2주차

그래프, DFS, BFS

<Graphs, DFS, BFS>

#그래프, DFS, BFS

<Graphs, DFS, BFS>

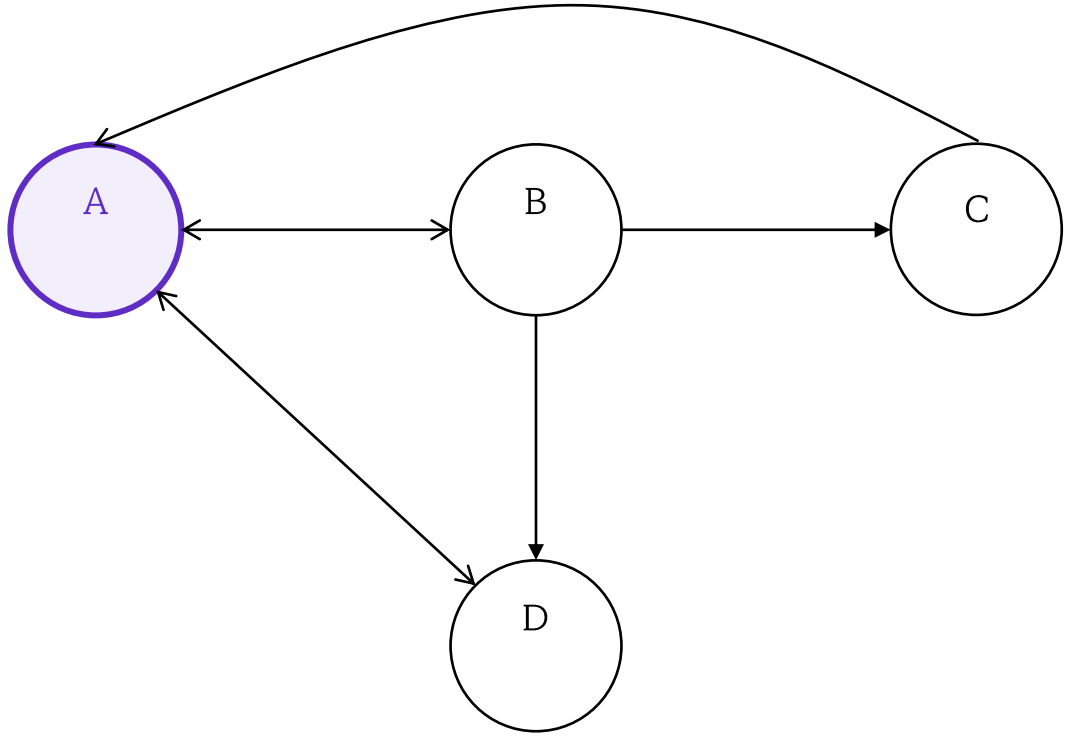
Graph (그래프) – 정점의 집합과 간선의 집합으로 정의되는 이산수학의 추상적인 구조

1. 일반적으로 정점은 어떠한 상태나 위치를 나타내고 간선은 각 정점을 이어주는 관계를 나타냅니다. 예를 들어 지하철 노선도 또한 하나의 그래프로 볼 수 있습니다. (정점 : 지하철 역, 간선 : 각 역 사이의 선로)
2. 그래프는 처음 보면 구조 자체를 받아들이기 어려울수 있지만 PS에서 굉장히 자주 문제가 출제되는 분야인 만큼 기초 문제부터 차근차근 풀어 익숙해지기 를 바랍니다.

#그래프, DFS, BFS

<Graphs, DFS, BFS>

(원은 정점 선은 간선)



#그래프, DFS, BFS

<Graphs, DFS, BFS>

그래프의 여러가지 종류:

1. 방향 그래프
2. 무방향 그래프
3. 가중 그래프
4. 이분 그래프

이것보다도 다양한 그래프들은 존재하지만 오늘은 이 4가지만 대표적으로 다뤄보도록 합시다!

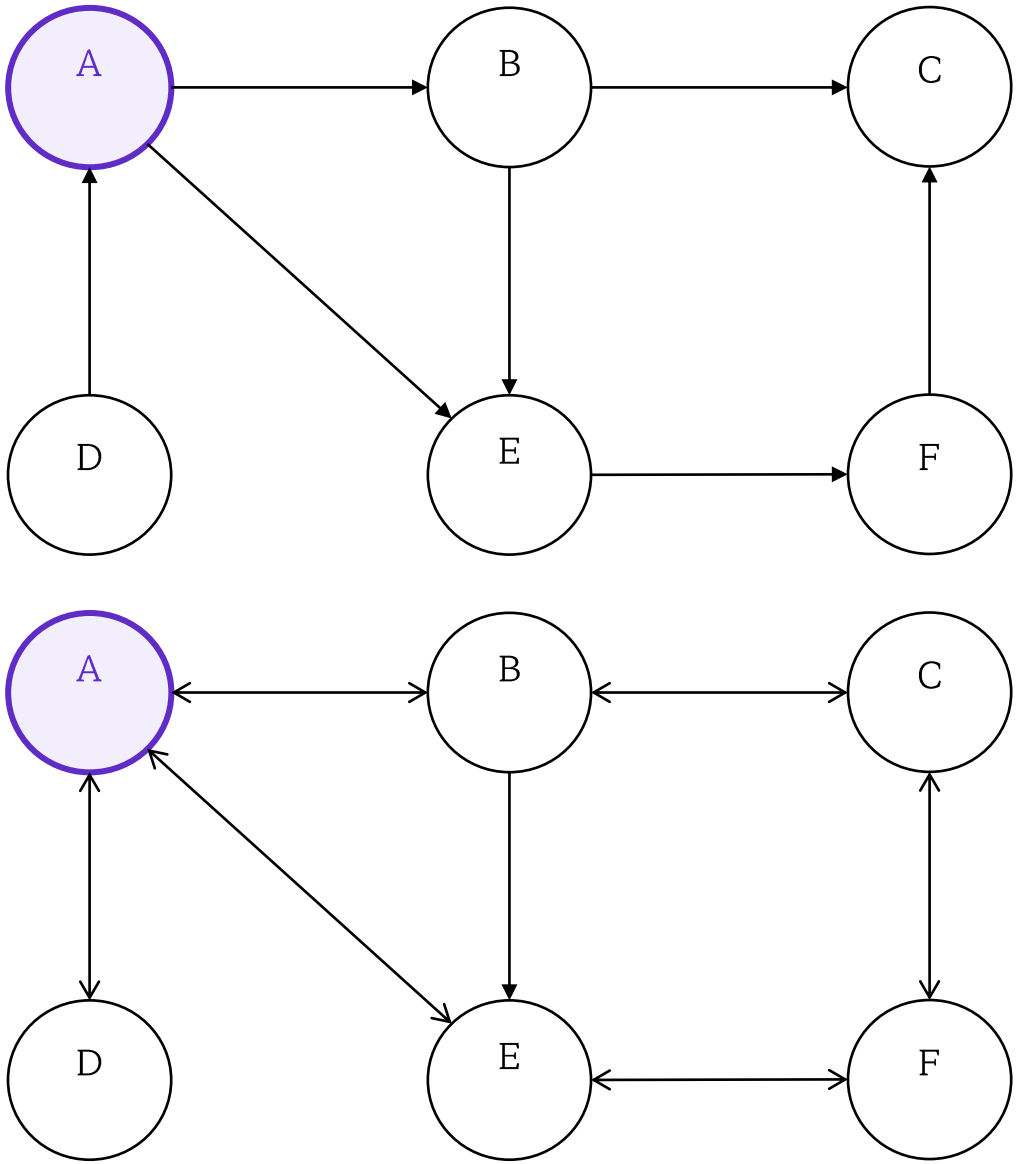
#그래프, DFS, BFS

<Graphs, DFS, BFS>

우선 방향, 무방향 (directed, undirected)그래프에 대해서 알아보도록 합시다.

방향 그래프는 간선에 방향이 정해져 있는 그래프를 의미하고 무방향 그래프는 간선에 방향이 정해져 있지 않은 그래프입니다. (하나의 간선으로 양방향으로 이동이 가능합니다)

오른쪽 그림들중 위가 방향, 아래가 무방향 그래프 입니다.

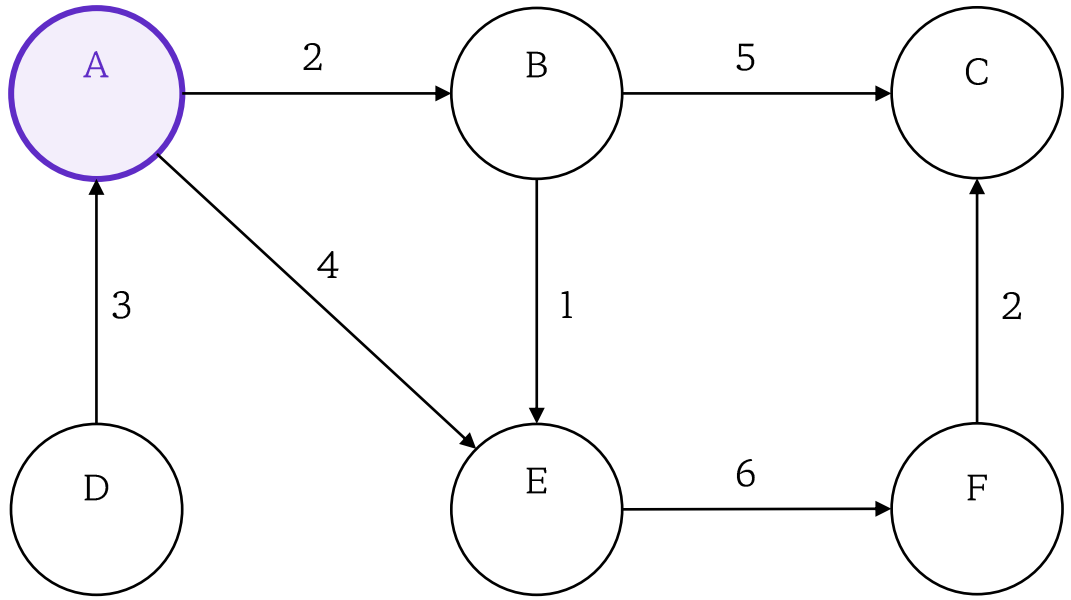


#그래프, DFS, BFS

<Graphs, DFS, BFS>

그 다음은 가중치 그래프에 대해서 알아봅시다. 가중치 그래프는 간선에 가중치 (cost 또는 weight) 이 존재하는 그래프 입니다.

우리가 실생활에서 쓰는 지도 또한 특정 지점들을 정점으로, 그 지점들 사이의 도로를 간선으로 생각하면, 그 도로들의 거리를 가중치로 나타내면 가중치 그래프로 모델링 해볼 수 있습니다.



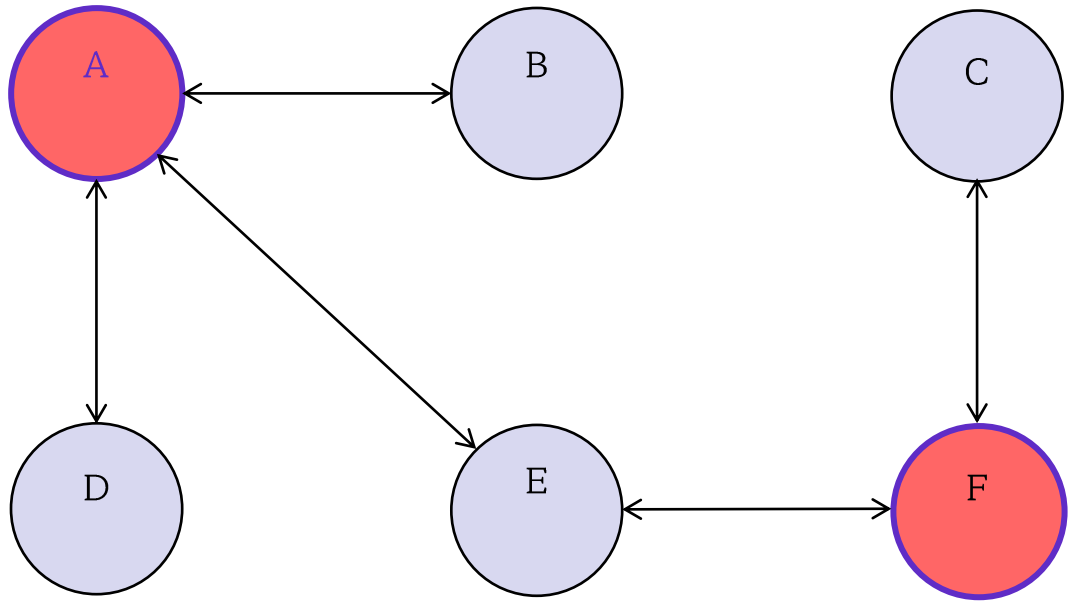
#그래프, DFS, BFS

<Graphs, DFS, BFS>

마지막은 이분그래프 (bipartite graph) 입니다.

이분 그래프는 정점들을 두 집합으로 나누었을 때 같은 집합의 정점끼리 는 간선으로 이루어지지 않게 할 수 있는 그래프를 의미합니다.

오른쪽 그래프는 이분 그래프입니다. 빨간색으로 칠해진 정점들과 파란색으로 칠해진 정점들을 각각의 집합으로 묶을 수 있습니다.



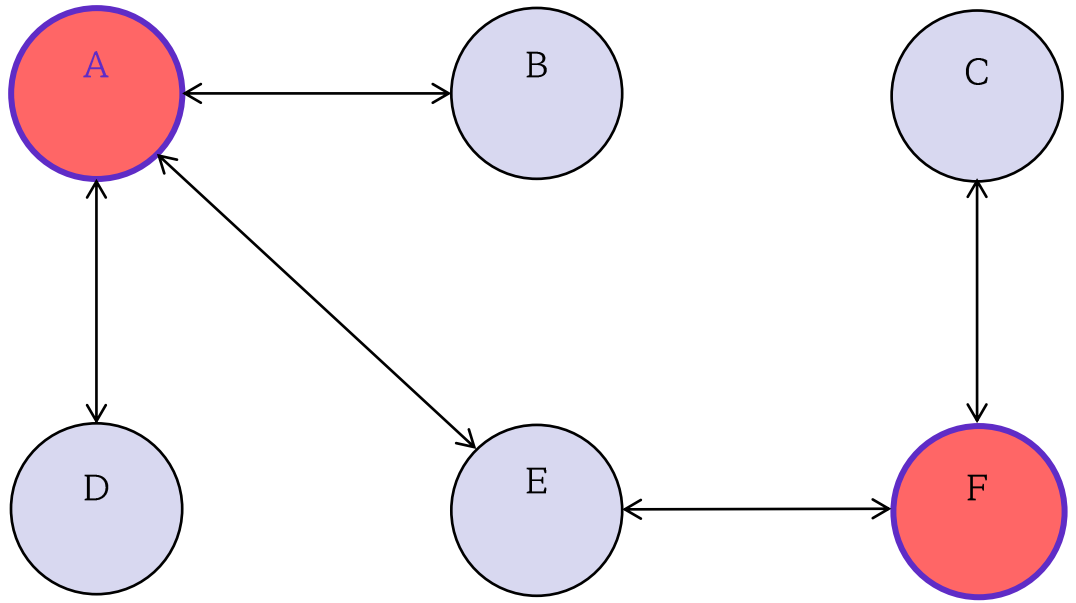
#그래프, DFS, BFS

<Graphs, DFS, BFS>

어떤 그래프가 이분 그래프인지 확인하는 방법은 생각보다 간단합니다. 특정 정점을 시작점으로 잡고 그 정점을 빨간색으로 칠해줍니다.

그 후 그 정점과 연결된 모든 정점들을 파란색으로 칠해줍니다. 그리고 그 파란색 정점들과 연결된 정점들을 빨간색으로 칠해주고… 이 과정을 반복합니다.

이분 그래프가 아니라면 이 과정에서 모순이 발생할겁니다.



#그래프, DFS, BFS

<Graphs, DFS, BFS>

그래프를 표현하는 방식으로는 대표적으로 2가지가 있습니다:
인접리스트 (Adjacency List) 와 인접행렬 (Adjacency Matrix)

1. 인접 리스트는 각각 정점들에 대해 간선으로 연결되어 있는 정점들을 각각의 리스트로 표현하는 방식입니다.
2. 인접 행렬은 i번 정점에서 j번 간선으로 가는 정점이 존재할 경우 행렬의 i번째 행 j번째 열의 값을 1, 존재하지 않을 경우 0 으로 설정해 표현하는 방식입니다.

#그래프, DFS, BFS

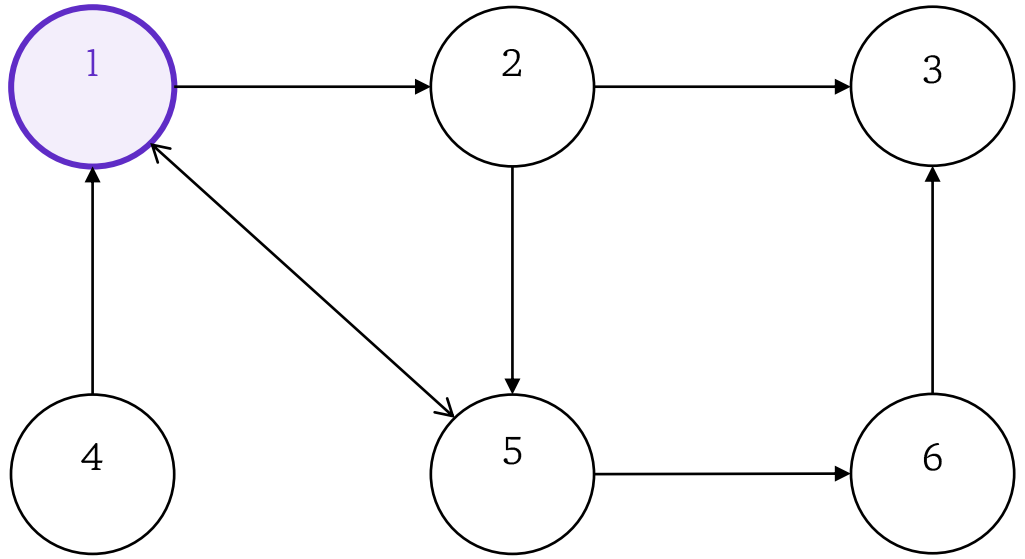
<Graphs, DFS, BFS>

<인접 리스트>

1	2, 5
2	3, 5
3	
4	1
5	1, 6
6	3

<인접 행렬>

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



#그래프, DFS, BFS

<Graphs, DFS, BFS>

C++로 인접 리스트를 어떻게 구현할수 있는지 알아봅시다.

인접 리스트의 구현 자체는 한줄로 충분합니다. 최대 정점 개수 만큼의 정수형 vector 배열을 만들어 주면 됩니다.

정점 u에서 정점 v로 가는 간선을 추가하기 위해서는 u번째 vector에 v를 추가해주시면 됩니다.

```
vector<int> edges[200005];

int u, v;
cin >> u >> v;
edges[u].push_back(v);
```

#그래프, DFS, BFS

<Graphs, DFS, BFS>

다음은 인접 행렬의 구현을 알아봅시다.

인접 행렬의 구현도 간단히 2차원 배열 하나로 끝납니다.

정점 u에서 정점 v로 가는 간선을 추가하기 위해서는 u번째 행 v번째 열을 1로 설정해주시면 됩니다.

```
int graph[305][305];

int u, v;
cin >> u >> v;
graph[u][v] = 1;
```

#그래프, DFS, BFS

<Graphs, DFS, BFS>

그래프에 대해 기본적인 지식을 배웠다면
"그래프 탐색" 알고리즘들에 대해 배울 차례입니다.

1. 그래프 탐색 알고리즘에는 다양한 종류가 있지만 오늘은 그중에서 가장 기본이라고 할 수 있는 DFS와 BFS에 대해서 배워봅시다.
2. 그래프 탐색은 최단 경로 문제, SCC 등 다양한 문제들에서 사용됩니다. 문제가 자주 출제되는 분야인 만큼 연습문제를 통해 익숙해지는 것이 중요합니다.

#그래프, DFS, BFS

<Graphs, DFS, BFS>

DFS/Depth-First-Search - 깊이 우선 탐색

1. 깊이 우선 탐색은 그래프에서 한 가지로 최대한 깊이 들어가서 탐색한후 더 이상 탐색할 수 없을 경우 돌아와서 다음 가지를 탐색하는 방식입니다.

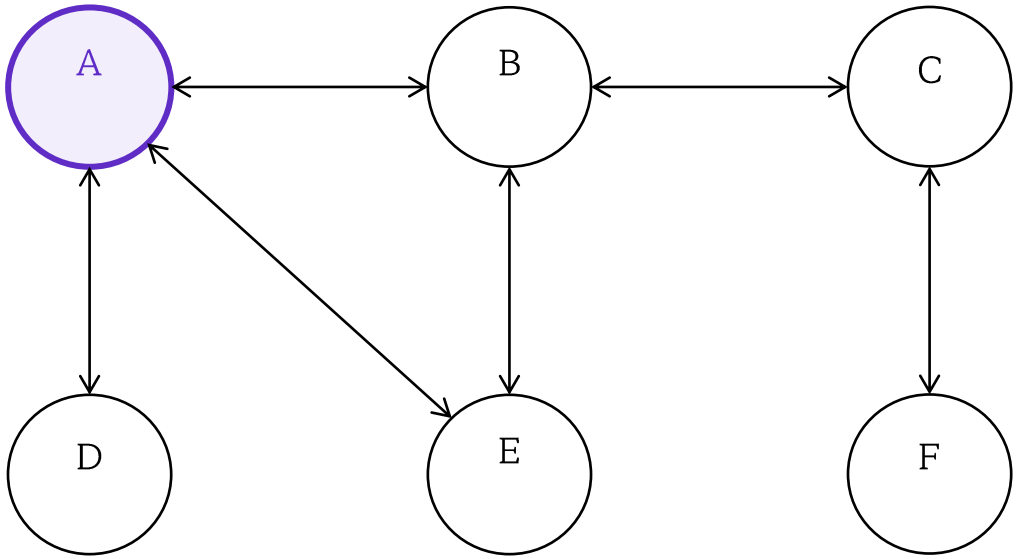
#그래프, DFS, BFS

<Graphs, DFS, BFS>

오른쪽 무방향 그래프를 가지고 한번 이해해보도록 합시다.

엄밀성을 위해 시작 정점은 A이며 탐색할수 있는 정점이 여러 개일 경우 사전순으로 가장 빠른 정점을 선택하며 한번 방문한 정점은 다시 방문하지 않는다고 합시다.

이 경우 방문 순서는 A B C F E D 가 됩니다.



#그래프, DFS, BFS

<Graphs, DFS, BFS>

이제 실제로 DFS를 구현해볼 차례입니다. DFS에는 크게 두가지 구현이 있습니다.

1. 스택을 이용한 비재귀적 구현
2. 재귀함수를 이용한 재귀적 구현

재귀함수를 사용한 구현이 더 짧고 직관적이라고 생각하므로 오늘은 재귀적 구현만 다뤄봅시다.

비재귀 구현은 추가 자료를 올리도록 하겠습니다.

```
bool vis[1005];

vector<int> edges[1005];

void dfs(int cur){
    vis[cur]=1;
    cout << cur << ' ';
    for(int& nxt: edges[cur]){
        if(vis[nxt]) continue;
        dfs(nxt);
    }
}
```

#그래프, DFS, BFS

<Graphs, DFS, BFS>

오른쪽은 간단한 재귀 dfs구현입니다. dfs로 방문한 순서대로 정점 번호를 출력합니다.

방문했던 정점은 다시 방문하지 않기 위해 vis라는 이름의 bool배열을 만들어줍니다. vis[i]의 값이 true면 i번째 정점을 이미 방문했다는 뜻입니다.

이제 천천히 dfs함수를 살펴보도록 합시다. 이 함수의 인자는 현재 정점 번호 (cur) 입니다. 우선 현재 방문한 정점을 방문했다고 기록을 해줍니다. 그 후 그 정점 번호를 출력합니다.

```
bool vis[1005];

vector<int> edges[1005];

void dfs(int cur){
    vis[cur]=1;
    cout << cur << ' ';
    for(int& nxt: edges[cur]){
        if(vis[nxt]) continue;
        dfs(nxt);
    }
}
```

#그래프, DFS, BFS

<Graphs, DFS, BFS>

그 후 그 정점과 연결된 정점을 확인합니다.

만약 이미 방문한 정점이라면 그 가지는 더 이상 탐색할수 없으므로 다른 정점을 확인합니다.

방문하지 않은 정점이라면 그 가지로 더 깊이 가기 위해 재귀 호출 인자로 그 정점 번호를 넘겨줍니다.

```
bool vis[1005];

vector<int> edges[1005];

void dfs(int cur){
    vis[cur]=1;
    cout << cur << ' ';
    for(int& nxt: edges[cur]){
        if(vis[nxt]) continue;
        dfs(nxt);
    }
}
```

#그래프, DFS, BFS

<Graphs, DFS, BFS>

BFS/Breadth-First-Search – 너비 우선 탐색

1. 너비 우선 탐색은 시작 정점으로 부터 가장 인접한 정점을 먼저 탐색하는 방식입니다.
2. 거리에 따라 단계별로 그 단계에 속하는 정점들을 방문한다고 생각하면 이해하기 편할 수 있습니다.

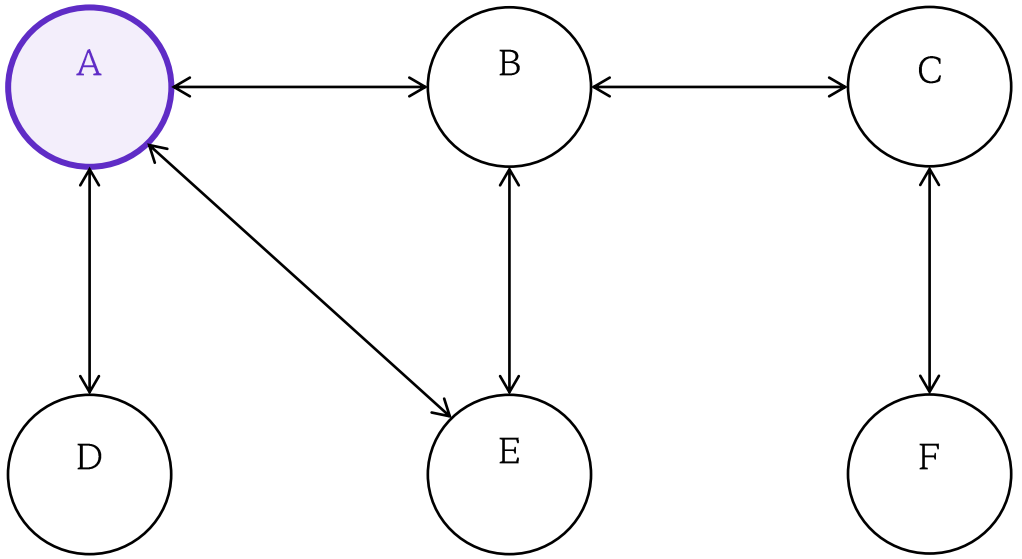
#그래프, DFS, BFS

<Graphs, DFS, BFS>

이번에도 오른쪽 무방향 그래프를 가지고 한번 이해해보도록 합시다.

엄밀성을 위해 시작 정점은 A이며 탐색할수 있는정점이 여러 개일 경우 사전순으로 가장 빠른 정점을 선택하며 한번 방문한 정점은 다시 방문하지 않는다고 합시다.

이 경우 방문 순서는 A B D E C F 가 됩니다.



#그래프, DFS, BFS

<Graphs, DFS, BFS>

이번에는 BFS를 구현해봅시다. DFS와 다르게 BFS는 재귀적으로 구현하지 않고 queue를 사용해 비재귀적으로 구현하는 방식이 좋습니다.

queue가 FIFO (First-In-First-Out) 형태의 자료구조임을 생각해보면 작동 방식을 이해하기 어렵지 않을 것입니다.

```
bool vis[1005];

vector<int> edges[1005];

void bfs(int start){
    queue<int>q;

    vis[start]=1;
    q.push(start);

    while(!q.empty()){
        int cur= q.front();
        cout << cur << ' ';
        q.pop();

        for(int& nxt:edges[cur]){
            if(vis[nxt]) continue;

            vis[nxt] = true;
            q.push(nxt);
        }
    }
}
```

#그래프, DFS, BFS

<Graphs, DFS, BFS>

queue에 시작 정점을 넣어줍니다.

그 후 queue가 빌 때 까지 다음 슬라이드에서 설명할 과정을 반복합니다.

```
bool vis[1005];

vector<int> edges[1005];

void bfs(int start){
    queue<int>q;

    vis[start]=1;
    q.push(start);

    while(!q.empty()){
        int cur= q.front();
        cout << cur << ' ';
        q.pop();

        for(int& nxt:edges[cur]){
            if(vis[nxt]) continue;

            vis[nxt] = true;
            q.push(nxt);
        }
    }
}
```

#그래프, DFS, BFS

<Graphs, DFS, BFS>

Queue가 빌 때 까지 (모든 정점을 방문 했을때 까지) 다음을 반복합니다.

1. queue의 맨 앞에 정점을 현재 방문한 정점으로 저장하고 그 번호를 출력합니다.
2. 그 정점과 연결된 정점들 중 아직 방문하지 않은 정점들을 queue에 넣어줍니다.
3. 그 정점과 연결된 정점들중 아직 방문하지 않은 정점들을 queue에 넣어줍니다.
4. 1~3을 queue가 빌 때 까지 반복합니다. queue가 비었다는 것은 더 이상 방문할수 있는 정점이 없다는 뜻이기 때문입니다.

```cpp
bool vis[1005];

vector<int> edges[1005];

void bfs(int start){
    queue<int>q;

    vis[start]=1;
    q.push(start);

    while(!q.empty()){
        int cur= q.front();
        cout << cur << ' ';
        q.pop();

        for(int& nxt:edges[cur]){
            if(vis[nxt]) continue;

            vis[nxt] = true;
            q.push(nxt);
        }
    }
}
```

#그래프, DFS, BFS

<Graphs, DFS, BFS>

DFS 와 BFS 의 시간복잡도 – O(V + E)
V는 정점의 개수 E는 간선의 개수

1. 모든 정점을 한번씩 방문하기 때문에 V
2. 모든 정점에서 연결된 간선을 한번씩 체크하기 때문에 (방문 여부 체크 과정에서) E

#그래프, DFS, BFS

<Graphs, DFS, BFS>

DFS와 BFS (BOJ #1260)

<문제 설명>

- 정점 N개와 간선 M개로 이루어진 그래프가 주어진다.
- 시작 정점이 주어진다
- 그래프를 DFS와 BFS로 각각 탐색한 결과를 출력한다.

<제약 조건>

- $1 \leq N \leq 1{,}000$
- $1 \leq E \leq 10{,}000$

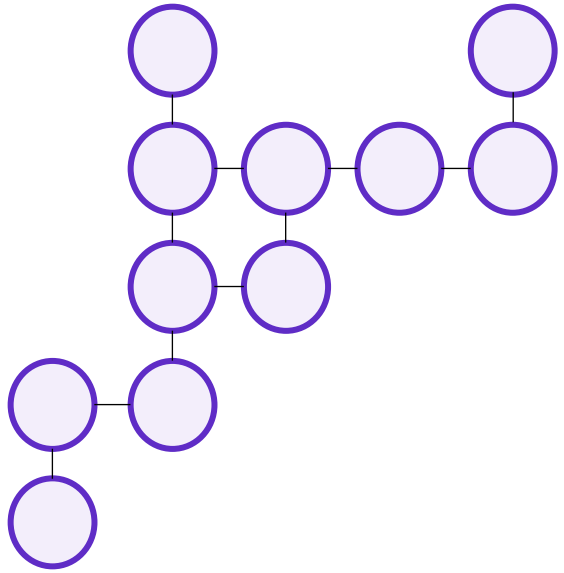
#그래프, DFS, BFS

<Graphs, DFS, BFS>

마지막으로 그래프 탐색을 이용하는 특정 분류의 문제들에 대해 알아보도록 합시다.
대기업 코딩 테스트에도 자주 출제되고 PS에도 비슷한 분류의 문제가 많기에 특별히 다뤄보도록 합시다.

2차원 그리드를 그래프로 보고 시작점에서 종착점까지 최단 경로를 구하는 문제입니다.
우선 그리드에서 이동할 수 있는 칸을 1 이동할 수 없는 칸을 0으로 표현하고 그리드의 인접한 칸으로 이동할 수 있다고 하면 오른쪽 위의 그리드는 그 아래의 그래프로 표현할 수 있습니다.

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



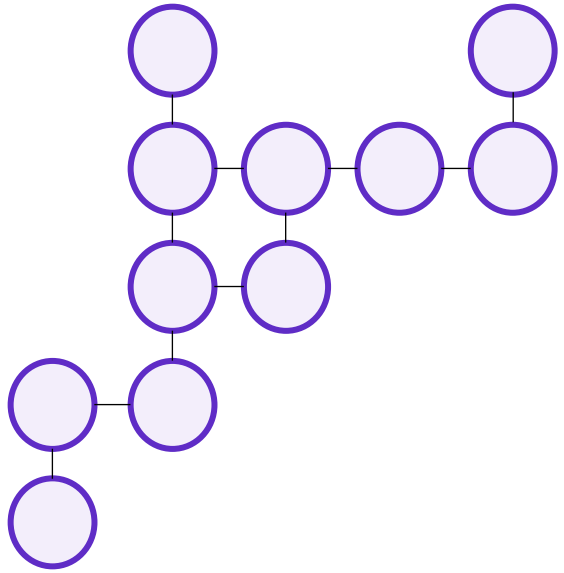
#그래프, DFS, BFS

<Graphs, DFS, BFS>

그 후 DFS나 BFS를 사용해 그래프를 탐색을 해주며 다음 방문한 정점까지의 거리를 현 정점의 거리 + 1 로 갱신해줍니다.

거리를 관리하는 방법은 따로 배열을 만들어 그 배열의 i행 j열의 값을 그리드의 i행 j열의 칸까지의 거리로 저장하면 됩니다.

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



#그래프, DFS, BFS

<Graphs, DFS, BFS>

1 미로 탐색 (BOJ #2178)

<문제 설명>

- 가로 M 세로 N 길이의 그리드가 주어진다
- 시작점은 1, 1로 고정되고 끝점은 N, M으로 고정될때 미로를 탈출하기 위한 최단거리를 출력한다

<제약 조건>

- $2 \leq M \leq 100$
- $2 \leq N \leq 100$

#연습 문제 도전

1 그림 (BOJ #1926)

그래프에 지친 우리 한번 예술 감각을 키워볼까요

4 이분 그래프 (BOJ #1707)

이분 그래프가 뭐더라… 슬라이드 앞에 있던거 같기도 하고…

3 벽 부수고 이동하기 (BOJ #2206)

사실 차원이 추가 된다는 것은 공간적인 의미만 가지는 것이 아닌 또다른 상태의 변화가 변수로 추가된다는 의미일수도 있습니다

5 토마토 (BOJ #7576)

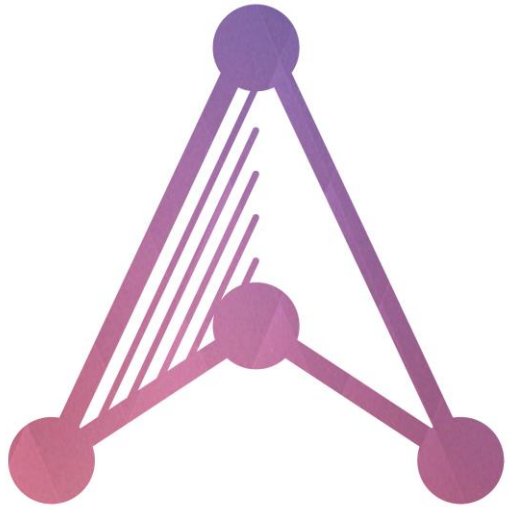
시작 정점이 여러 개인 상황에서의 BFS는 어떻게 처리해야 할까요? (11차원 버전이 있다는 괴담이…)

4 로하의 농사 (BOJ #26153)

HCPC 기출을 풀어봅시다!

5 즉흥 여행 (Easy) (BOJ #26146)

HCPC 기출 2, SCC라는 이상한걸 사용하면 더 쉽게 풀 수 있지만 DFS만으로도 충분히 해결할 수 있으니 고민해보는 시간을 가지셨으면 합니다!



ALOHA

The algorithm club.