ICPC Sinchon

2022 Winter Algorithm Camp

8회차. 그래프/그래프 탐색

서강대학교 김성현

2022 Winter Algorithm Camp
8회차. 그래프/그래프 탐색

목차

강의에서 전하고 싶은 것

*** 그래프에 대한 직관적인 이해**

- 그래프의 기본을 익히는 것만으로도 solved.ac 기준 실버 상위~골드 하위 문제를 꽤 풀 수 있다
- 그러나 그래프는 그 정도 티어를 훨씬 뛰어넘는 가치를 가짐
- 다음 강의의 주제인 트리 또한 그래프 응용의 일종이며, 그 외에도 수많은 응용이 있다
- 응용 중 유명한 알고리즘이나 자료구조만 해도 상당하다
- 이 강의에서 전해주고 싶은 것은 그래프의 기초와 한두 가지 응용
- 그래프를 다루는 법은 더 고급의 알고리즘을 사용할 때도 중요하다
- 많은 문제를 풀면서 감을 익히는 것도 중요(어쩌면 제일 중요)
- 문제의 상황을 그래프로 생각할 수 있어야 한다

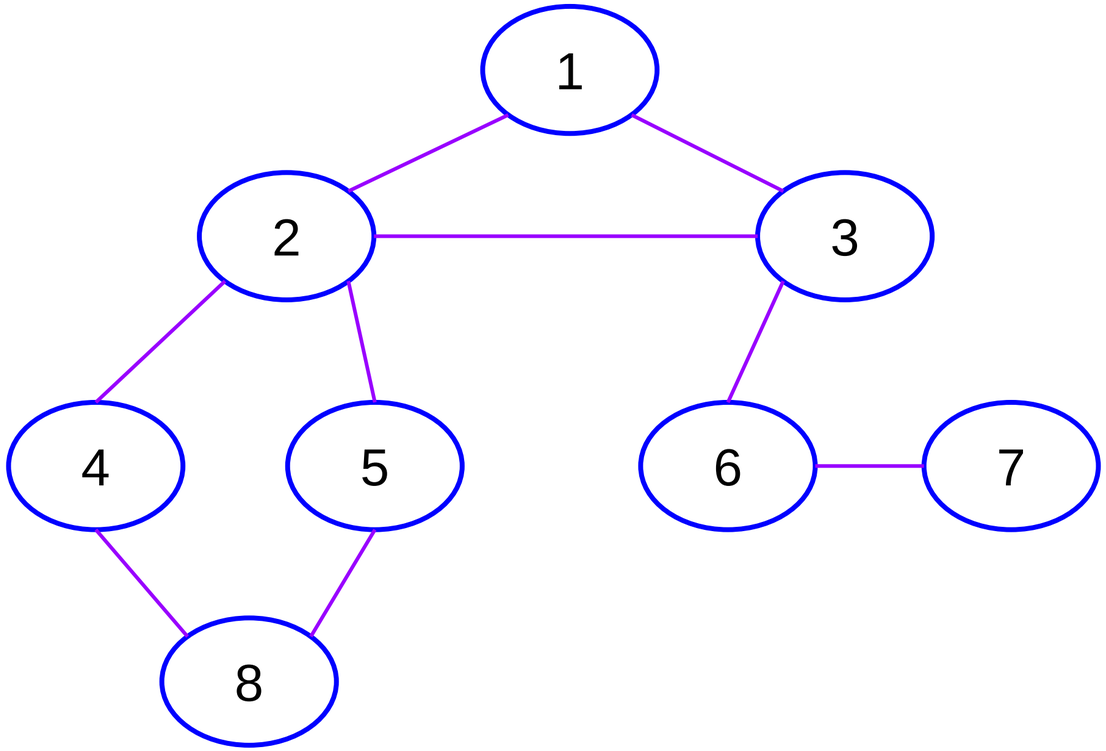
그래프 - 개념

* 그래프란 무엇인가?

- 그래프의 엄밀한 정의는 정점(vertice)의 집합 V(not empty)와 간선(Edge)의 집합 E로 이루어진다.
- 보통 (V,E) 로 표기
- 각각의 간선은 두 정점의 쌍으로 나타난다. (<a,b>)
- 간선에 방향이 있거나 가중치가 있거나, 자기 자신으로 가는 간선이 있는 등 다양한 속성이 부여되기도 한다
- 그러나 기본적으로는 어떤 정점들의 집합과 정점들 간의 연결 관계를 나타내는 간선 집합으로 이루어진 것이 바로 그래프
- 정점이 될 수 있는 것에 특별한 제한은 없다

그래프 - 개념

*** 그래프란 무엇인가?**



- 정점이 될 수 있는 것에 딱히 제한은 없으나 편의상 정점 하나를 정수 하나에 대응시켜 표현하였다
- 정점은 {1,2,3,4,5,6,7,8}
- 간선은 {<1,2>, <1,3>, <2,3>, <2,4>, <2,5>, <3,6>, <4,8>, <5,8>, <6,7>}
- 간선에 방향, 가중치가 없는 경우

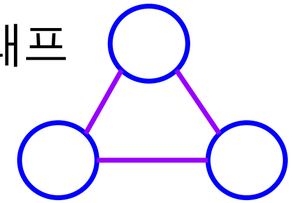
그래프 - 개념

* 그래프에서 쓰이는 용어

- **무방향 그래프** : 간선의 방향이 없음

  즉 a와 b 사이에 간선이 있으면 a->b, b->a 둘 다 가능
- **방향 그래프** : 간선의 방향이 있음

  화살표 방향에 따라 갈 수도 있고 안 갈 수도 있다
- **인접** : 정점 A와 정점 B가 간선 하나로 연결되어 있다. 두 정점이 양끝인 간선이 있는 것
- **사이클** : 한 정점에서 시작해서 다시 그 정점으로 돌아오는 경로. A->B->C->A 같은 것
- **가중치 그래프** : 간선들에 가중치가 부여된 그래프(이후 강의에서 재등장 예정)
- **연결 그래프** : 정점들이 모두 이어져 있는 그래프



2022 Winter Algorithm Camp

그래프 - 개념

*** 여러 가지를 그래프로 생각할 수 있다**

- 가령 지도에서 길을 찾는 것도 지도의 특정 위치들을 정점으로 잡고 정점들 간의 최단경로를 찾는 것으로 생각할 수 있다

- 어떤 것들의 연결 관계를 나타내는 것
- 정점이 있고 그걸 연결하는 간선들로 이루어져 있는 게 그래프라는 것만 어렴풋이 이해해도 된다

그래프 - 표현

*** 그래프를 코드로 어떻게 표현할 것인가?**

- 정점들은 보통 정수 하나로 표현되거나 구조체로 나타낼 수 있다
- 따라서 정점은 표현하기 어렵지 않다
- 그러면 정점들 간에 연결 관계를 나타내는 간선은 어떻게 표현할 것인가?

*** 간선이 표현해야 할 것은 무엇인가?**

- 각각의 정점이 어떤 다른 정점과 연결되어 있는지를 표현해야 한다
- 즉 임의의 두 정점 간에 대응 관계만 표현할 수 있으면 된다
- 인접 리스트, 인접 행렬

2022 Winter Algorithm Camp

그래프 - 표현

* 인접 리스트로 그래프를 표현

- 각 정점들마다 연결되어 있는 정점을 저장해 놓자
- 벡터 배열을 이용한다. 만약 a 정점이 b 정점과 연결되어 있다면 배열의 a 인덱스 벡터에 b가 들어 있음
- `vector<int> adj[100005];`
- adj[a].push_back(b);
- python의 경우 dict나 list 사용 가능
- 중요한 것은 각 정점마다 그 정점과 연결된 정점들을 담고 있는 컨테이너(vector, set etc.)를 대응시켜 주는 것

그래프 - 표현

* 인접 행렬로 그래프를 표현

- 각 정점의 연결 관계를 저장해 놓자
- 정점 개수가 n일 때 nxn 배열을 이용한다. a 정점과 b 정점이 연결되어 있을 때 adj[a][b]가 1이다
- `int adj[1005][1005];`
- adj[a][b] = 1;
- 두 정점을 알면 바로 그 두 정점이 연결되어 있는지를 알 수 있다
- 간선에 가중치 부여도 쉽게 가능. adj[a][b]=c; (c가 간선의 가중치) 를 하면 된다
- 하지만 공간복잡도가 n^2이기 때문에 정점 개수가 많아지면 쓰기 힘들다
- sparse table로 나타낼 수도 있지만 인접 리스트를 쓰는 경우가 더 많다

그래프 - 표현

*** 보통의 경우 인접 리스트를 많이 사용한다**

- 대부분의 문제들의 정점 개수가 꽤 많기 때문에 인접 리스트가 더 유리하다
- 구현도 더 편한 면이 있다
- 따라서 이 강의에서는 인접 리스트로 그래프를 표현한다고 생각하고 설명한다

그래프 - 깊이 우선 탐색

* 그래프의 순회

- 그래프는 어떤 상태들 간의 연결 관계를 나타낸 것
- 그래프를 순회하면서 어떤 조건을 만족하는 상태의 수를 세거나, 최단거리를 찾거나 그래프의 특정 요소를 업데이트하는 등 여러 가지 작업을 할 수 있기 때문에 그래프 순회는 중요하다

* 그래프의 두 가지 탐색 방식

- 깊이 우선 탐색과 너비 우선 탐색이 있다
- 먼저 배울 깊이 우선 탐색은 시작 정점에서 말 그대로 최대한 깊이 탐색해 나가는 것
- 이전 강의에서 다룬 스택 자료구조를 사용하므로 스택에 대한 이해가 필요

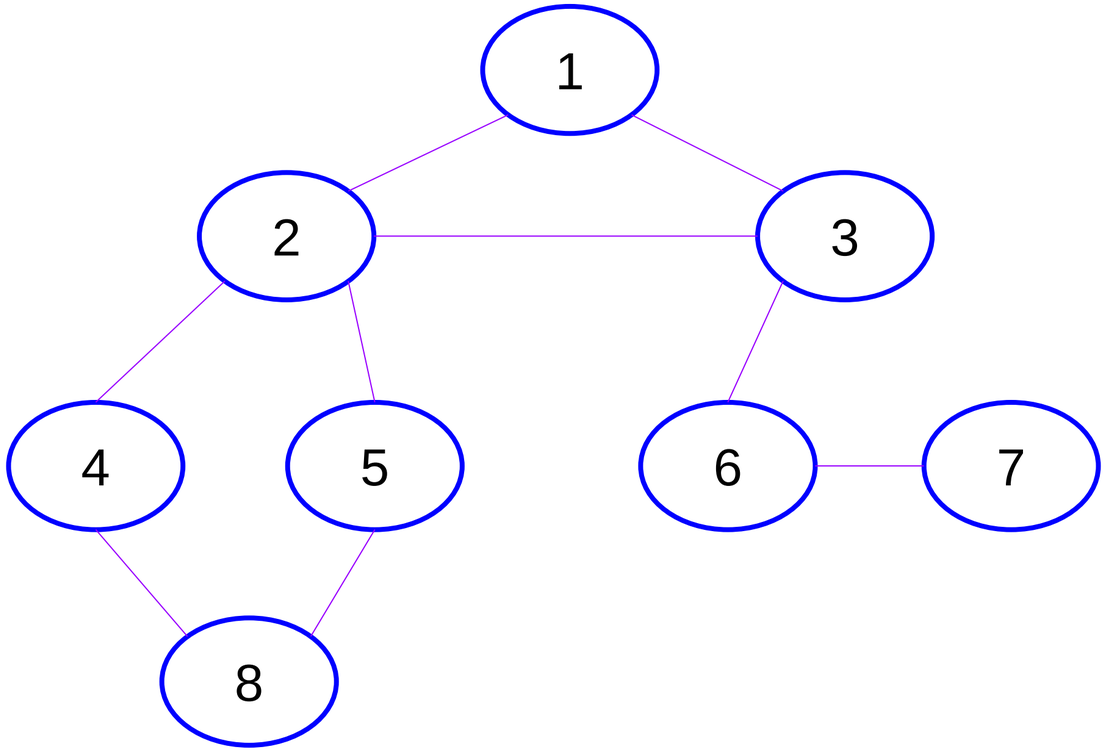
그래프 - 깊이 우선 탐색

* 그래프의 깊이 우선 탐색

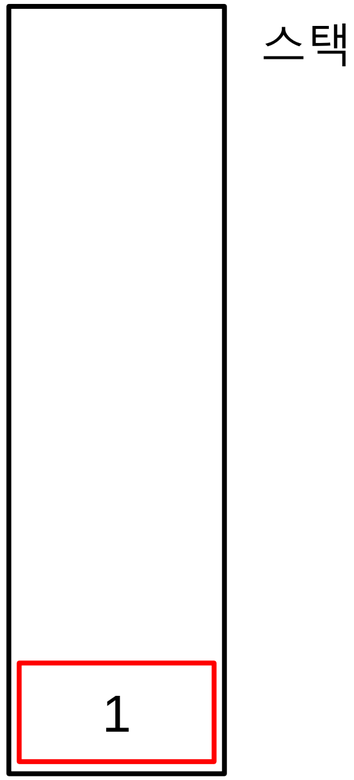
- Depth First Search의 약자로 DFS라는 말을 흔히 사용
- 시작 정점에서 시작해서 인접한 정점 중 하나를 택해서 방문한다
- 그 정점에 인접한 정점 중 아직 방문하지 않은 정점을 또 택하여 방문한다
- 만약 인접한 정점들을 다 방문했다면 이전 정점으로 계속 후퇴하면서, 아직 방문하지 않은 인접 정점이 있을 때 그 정점을 방문한다
- 스택의 끝에 있는 정점을 빼서 방문한다. 그리고 그 자식 정점들을 스택에 넣는 것을 반복한다

그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**

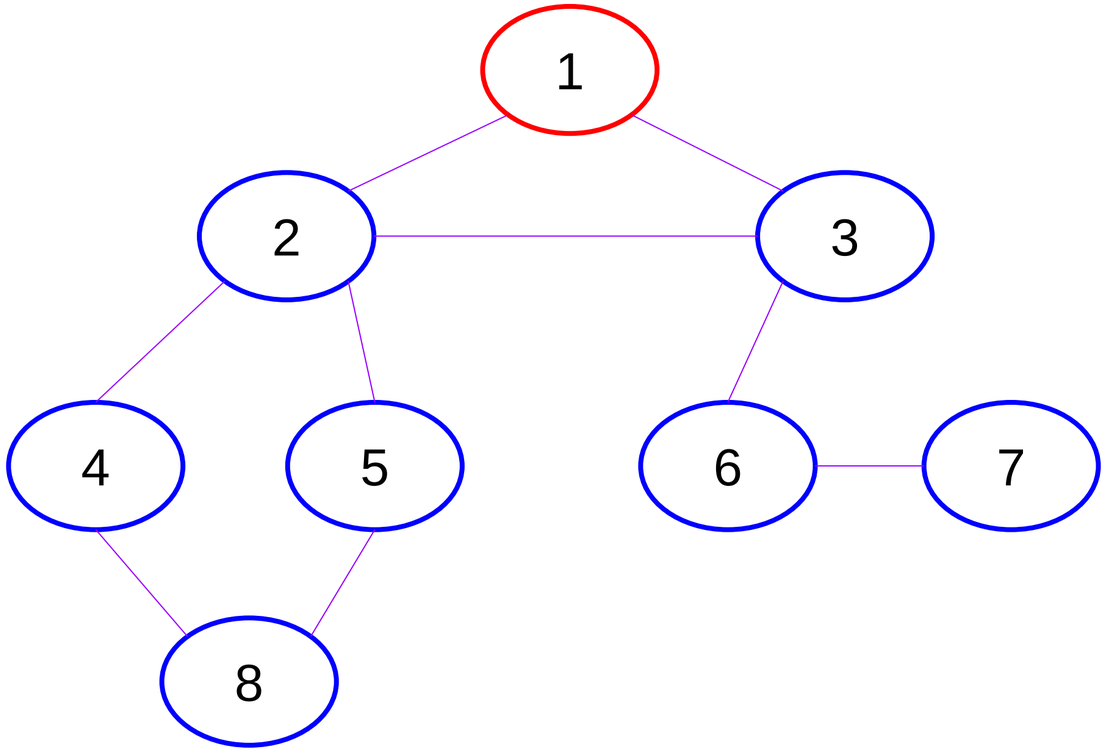


- 1번 정점에서 시작한다

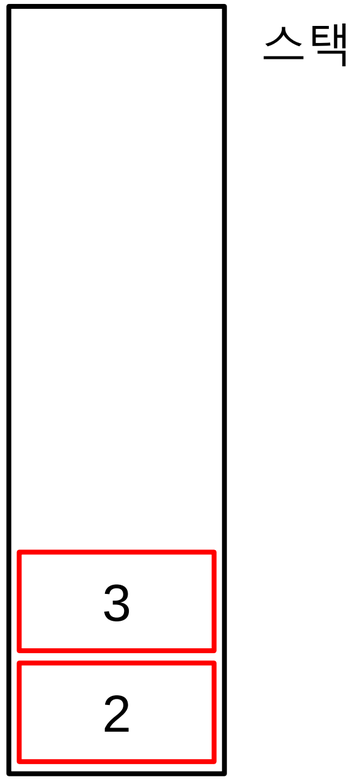


그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**

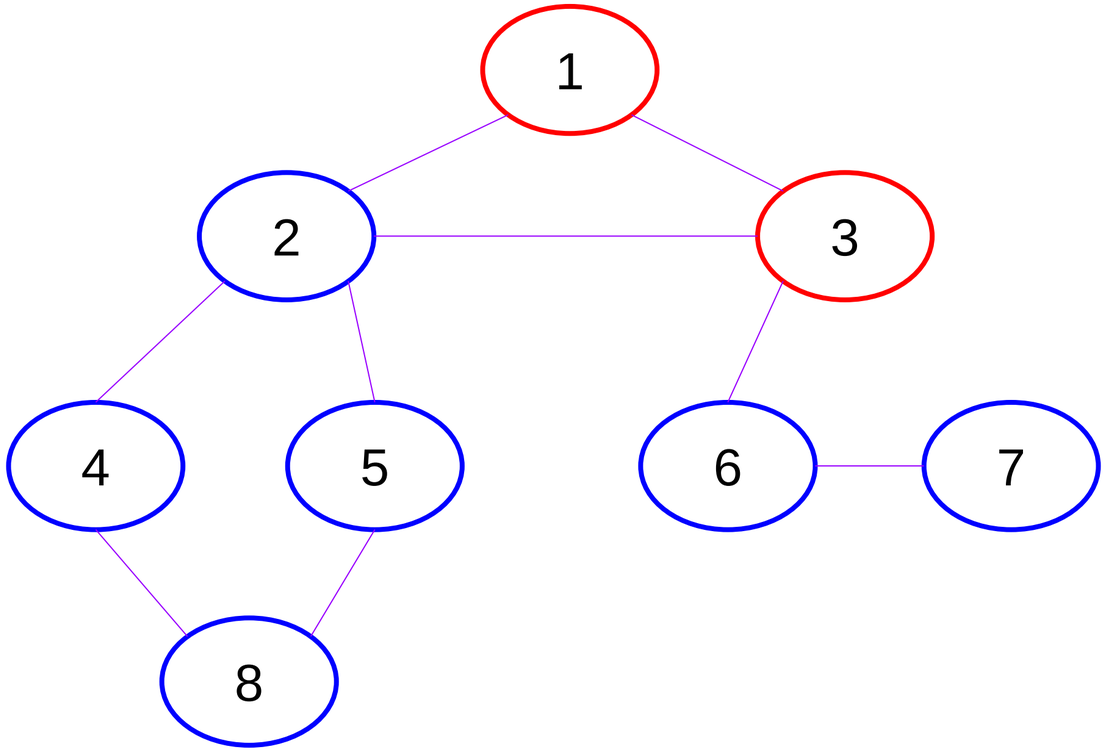


- 1번 정점을 스택에서 빼서 방문하고 자식들을 스택에 넣는다

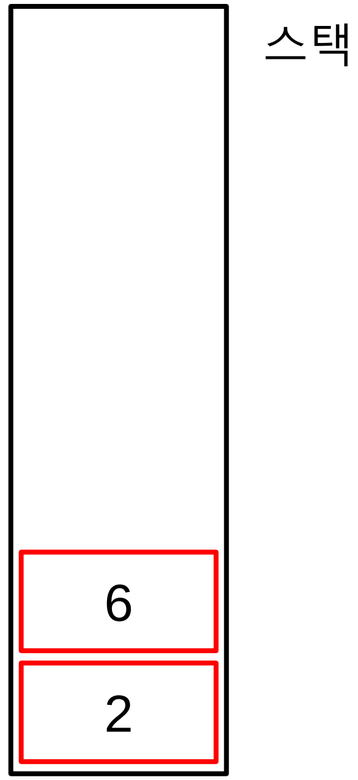


그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**

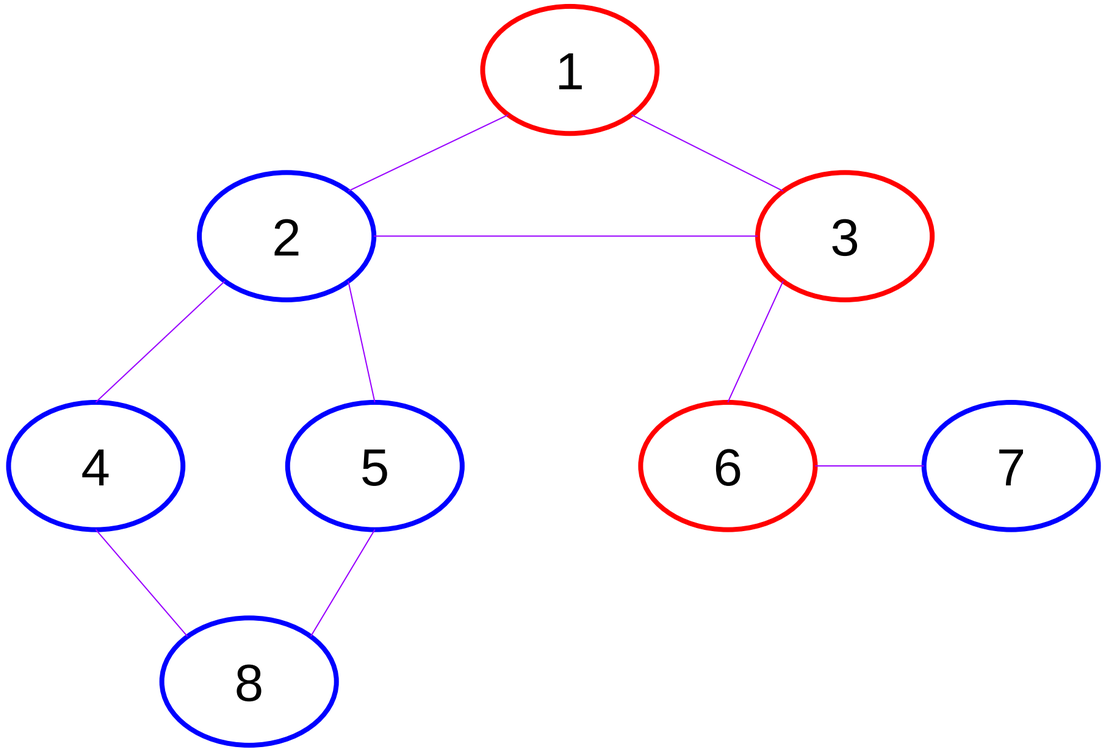


- 3번 정점을 스택에서 빼서 방문하고 아직 스택에 넣지 않은 3번의 자식들을 스택에 넣는다

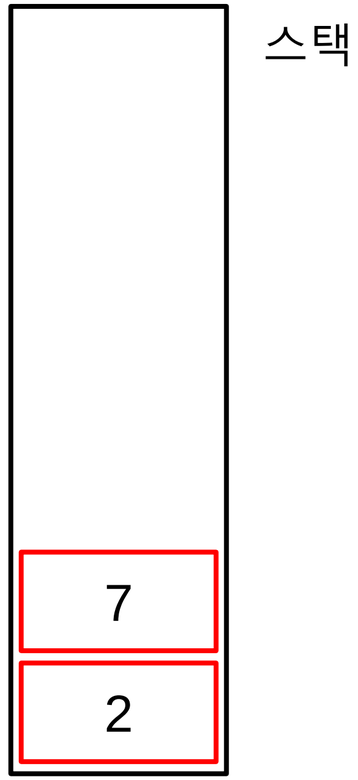


그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**

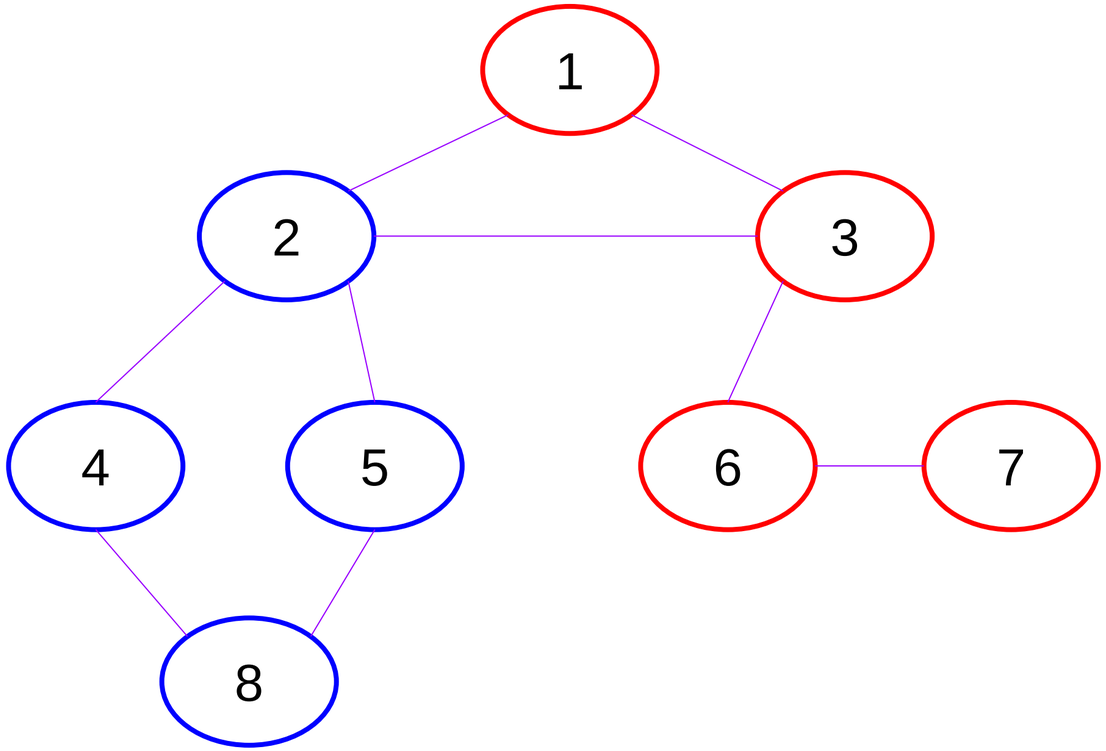


- 6번 정점을 스택에서 빼서 방문하고 아직 스택에 넣지 않은 6번의 자식들을 스택에 넣는다

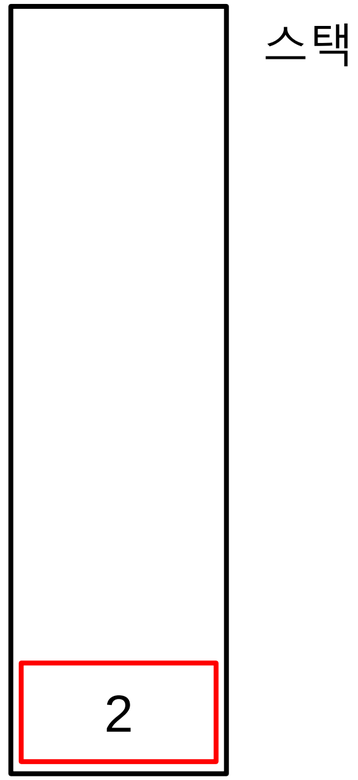


그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**

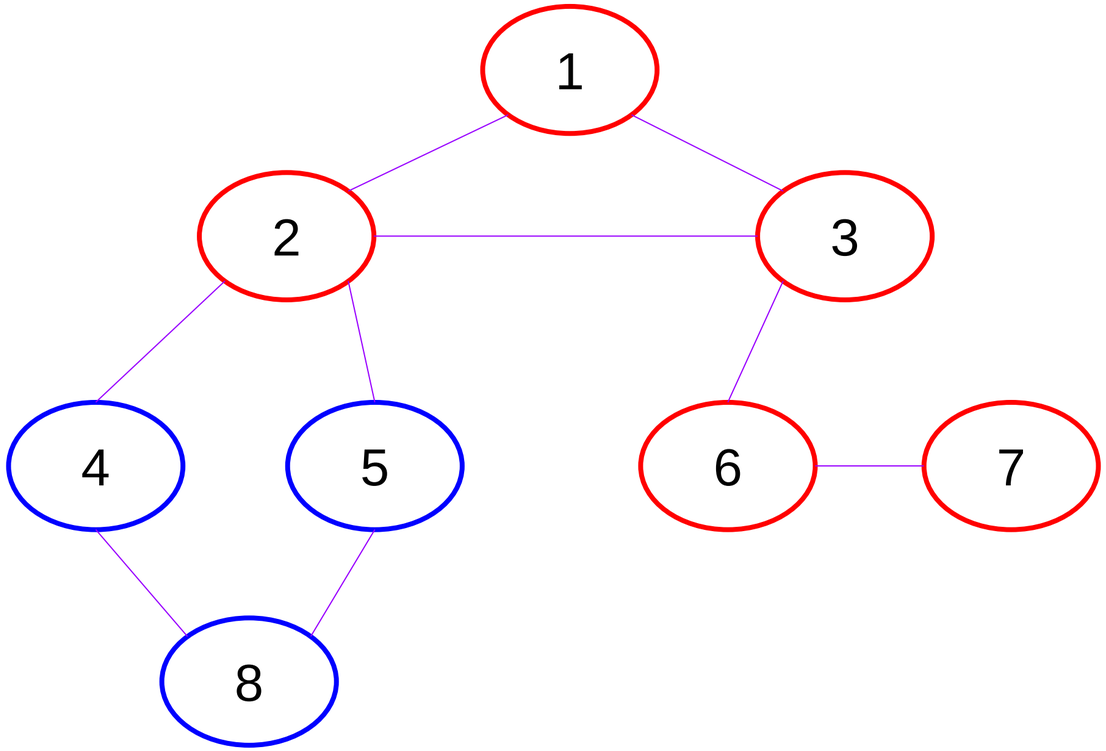


- 7번 정점을 스택에서 빼서 방문한다. 아직 스택에 들어간 적 없는 7번의 자식이 없으므로 그냥 넘어간다

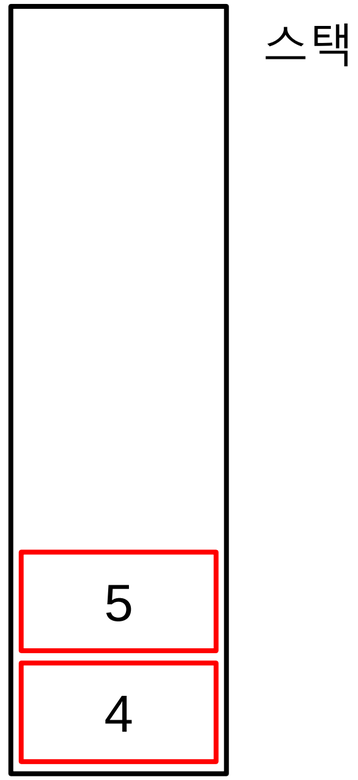


그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**

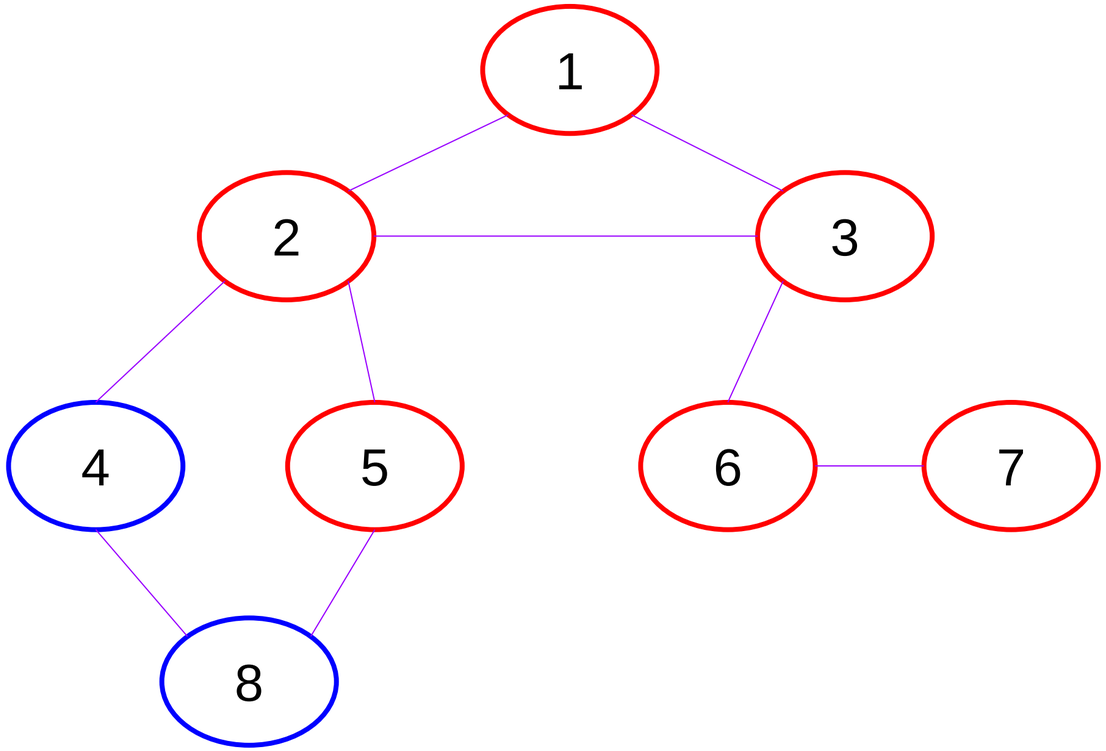


- 2번 정점을 스택에서 빼서 방문하고 아직 스택에 들어간 적 없는 자식들을 스택에 넣는다

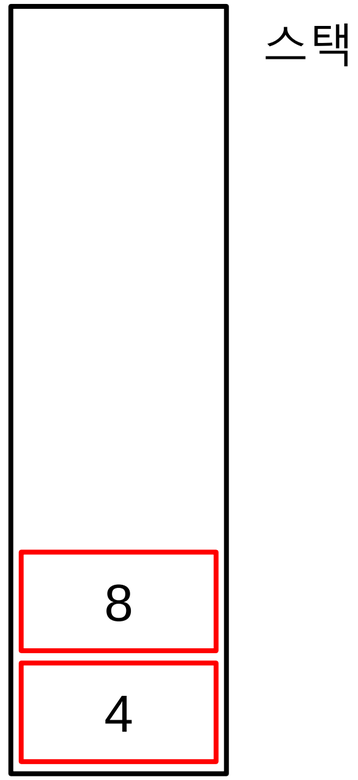


그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**

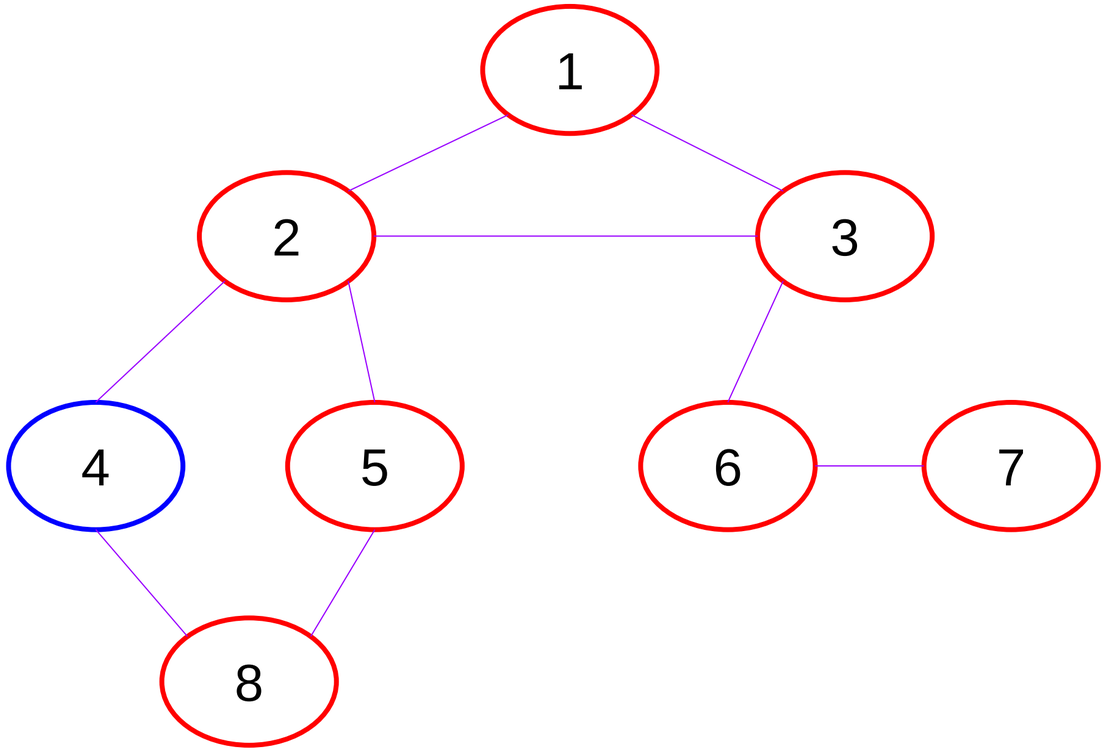


- 5번 정점을 스택에서 빼서 방문하고 아직 스택에 들어간 적 없는 자식들을 스택에 넣는다

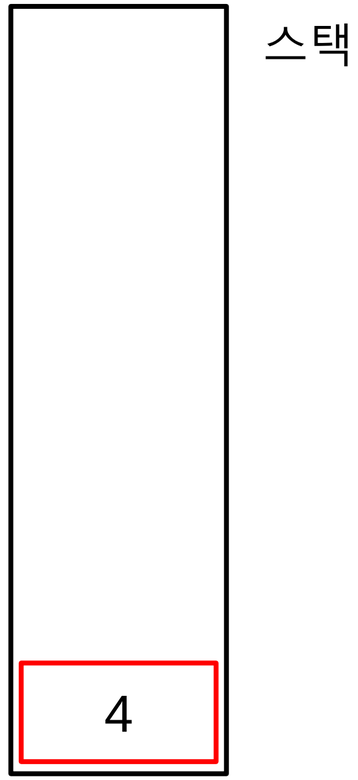


그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**



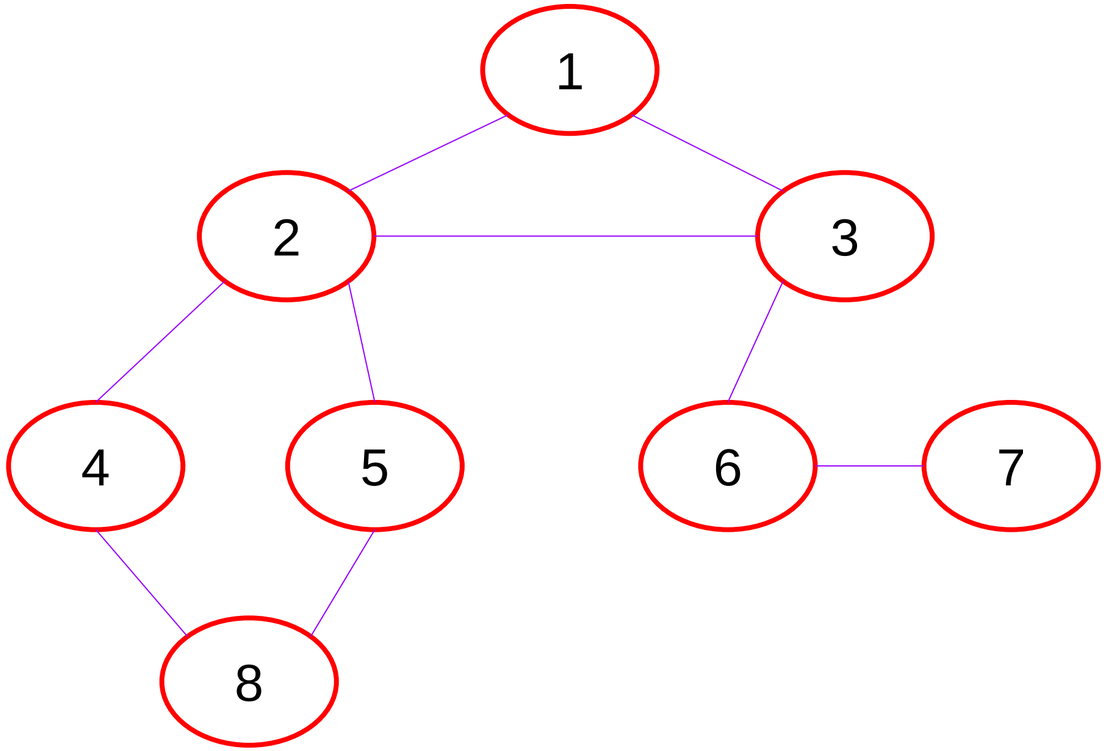
- 8번 정점을 스택에서 빼서 방문하고 아직 스택에 들어간 적 없는 8번의 자식이 없으므로 넘어간다



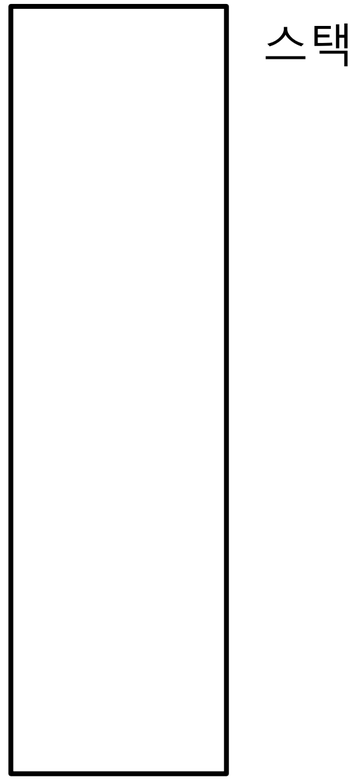
2022 Winter Algorithm Camp

그래프 - 깊이 우선 탐색

*** 그래프의 깊이 우선 탐색**



- 마지막으로 스택에 남아 있는 4번 정점을 빼서 방문하고 스택이 비었으므로 탐색을 종료한다



그래프 - 깊이 우선 탐색

* 깊이 우선 탐색 구현

- 설명과 그림에서 보았듯이 기본적으로 스택을 활용한다
- 재귀를 통해서 스택을 쌓을 수도 있고 스택 자료구조를 활용해서 구현할 수도 있다
- 각각 나름의 장점이 있다.
- 지금 중요한 것은 구현 방식보다는 순회의 구조를 이해하는 것

* 깊이 우선 탐색

- 방문한 노드의 자식을 방문하고, 또 그 자식을 방문하는 것을 반복하면서 깊게 탐색해 나감
- 그 경로에서 더 방문할 노드가 없을 경우 아까 방문하지 못했던 정점들로 되돌아옴
- 이 순서를 구현하기 위한 방식이 바로 스택

그래프 - 깊이 우선 탐색 : 구현

* 깊이 우선 탐색 구현 - 재귀

- 설명과 그림에서 보았듯이 기본적으로 스택을 활용한다
- 재귀를 통해서 스택을 쌓을 수도 있고 스택 자료구조를 활용해서 구현할 수도 있다
- 각각 나름의 장점이 있다.
- 지금 중요한 것은 구현 방식보다는 순회의 구조를 이해하는 것

```
vector<int> adj[1005];
int visited[1005]={0};

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

- 이미 방문한 정점이 아니라면 현재 정점을 방문한 후 그 자식 정점들을 타고 재귀적으로 내려간다

그래프 - 깊이 우선 탐색 : 구현

* 깊이 우선 탐색 구현 - 스택

- 설명과 그림에서 보았듯이 기본적으로 스택을 활용한다
- 재귀를 통해서 스택을 쌓을 수도 있고 스택 자료구조를 활용해서 구현할 수도 있다

```
vector<int> adj[1005];
int visited[1005]={0};

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]){continue;}
            visited[next]=1;
            //do something
            st.push(next);
        }
    }
}
```

- 현재 방문하는 정점의 자식 정점들 중 방문 상태가 아닌 걸 스택에 넣는다
- 스택의 역할을 할 수 있는 다른 stl들도 있다. list, deque, 심지어 vector도 가능하며, 배열을 이용해 구현한 야매 스택으로도 가능하다. 물론 링크드 리스트를 이용해 정석적으로 구현할 수도 있다

그래프 - 깊이 우선 탐색 : 구현

* 번외 - 스택의 배열 구현

- 앞 슬라이드에서, 배열을 이용한 야매(?) 스택으로도 DFS 구현이 가능하다는 언급을 했다
- 배열의 크기를 문제의 최대 입력 제한을 넘을 만큼 넉넉하게 잡아 두고, 스택의 상단을 표현하는 인덱스를 두면 스택의 모든 연산을 구현 가능하다
- 물론 그다지 좋은 방식은 아니다
- 어떤 스택을 쓰고, 재귀를 사용하고 안 사용하고가 아니라 스택의 삽입/삭제 구조와 그것이 DFS의 그래프 탐색 순서에 어떻게 작용하는지를 이해하는 게 구체적인 구현보다 훨씬 중요하다는 사실을 강

```
int array_stack[200005];
int stack_back_idx=0, stack_size=0;

void push(int data){
    //스택에 새로운 원소 삽입
    array_stack[stack_back_idx]=data;
    stack_back_idx++; stack_size++;
}
```

```
int pop(){
    //스택 최상단 pop
    if(stack_size==0){return 0;}
    // 스택 사이즈가 0이라 pop을 할 수 없을 시 0 반환
    stack_back_idx--; stack_size--;
    return 1; //pop 성공시 1 반환
}
```

```
int top(){
    //스택의 최상단 원소를 리턴
    if(stack_size==0){return -1;} //만약 최상단 원소 없으면 -1 리턴.
    //모든 스택의 원소는 0 이상이라 가정한다
    return array_stack[stack_back_idx-1];
}
```

그래프 - 깊이 우선 탐색 : 구현

* 번외 - 스택의 배열 구현

- 앞 슬라이드에서, 배열을 이용한 야매(?) 스택으로도 DFS 구현이 가능하다는 언급을 했다
- 배열의 크기를 문제의 최대 입력 제한을 넘을 만큼 넉넉하게 잡아 두고, 스택의 상단을 표현하는 인덱스를 두면 스택의 모든 연산을 구현 가능하다
- 물론 그다지 좋은 방식은 아니다
- 어떤 스택을 쓰고, 재귀를 사용하고 안 사용하고가 아니라 스택의 삽입/삭제 구조와 그것이 DFS의 그래프 탐색 순서에 어떻게 작용하는지를 이해하는 게 구체적인 구현보다 훨씬 중요하다는 사실을 강조하는 것
- 각자의 구현은 스스로가 많은 문제를 풀면서 본인이 가장 편하고 직관적인 스타일로 정립시켜 나가는 것
- 많은 알고리즘 대회가 팀 대회임을 고려하면 int a,b,c,e,d,f,g,...z; 만으로 모든 구현을 하는 수준은

그래프 - 너비 우선 탐색

* 너비 우선 탐색

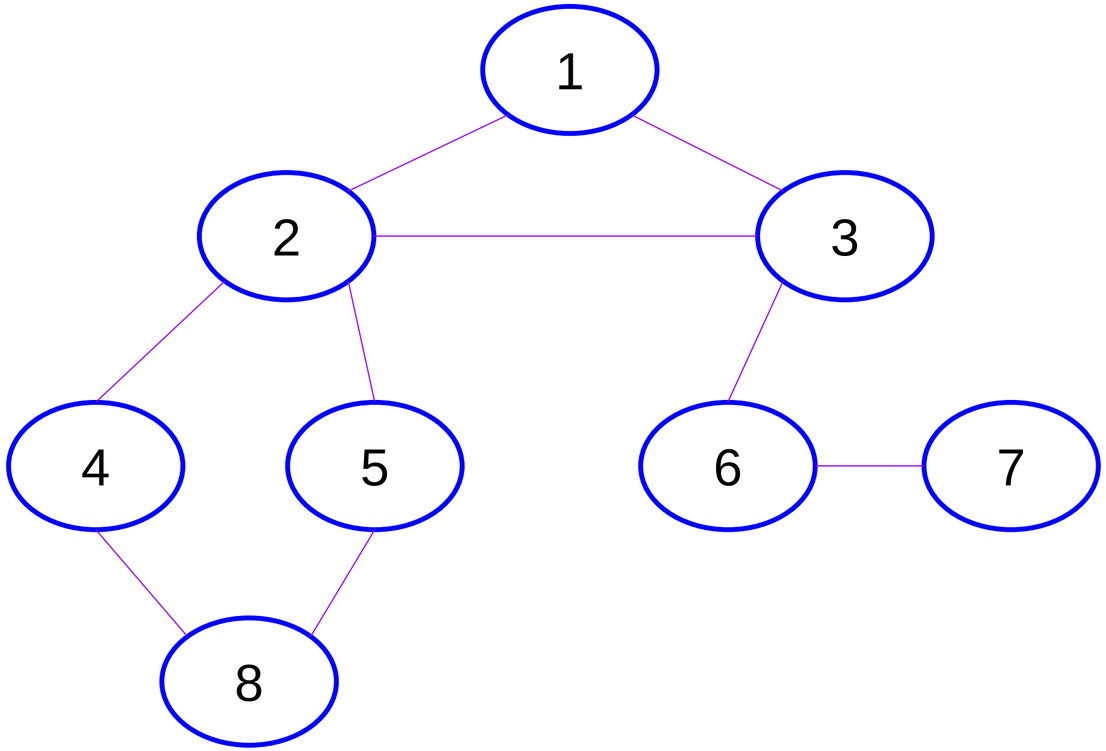
- Breadth First Search, BFS라고 더 많이 불린다
- 너비 우선 탐색은 시작 정점에서 시작해서 인접한 정점을 모두 탐색한 후 다음 단계로 나아간다
- 시작 정점에서 더 깊어지는 방향으로 일단 내려가 보는 깊이 우선 탐색과 달리 너비 우선 탐색은 한 단계 깊이를 모두 탐색한 후 다음 깊이로 내려감

* 너비 우선 탐색의 구현

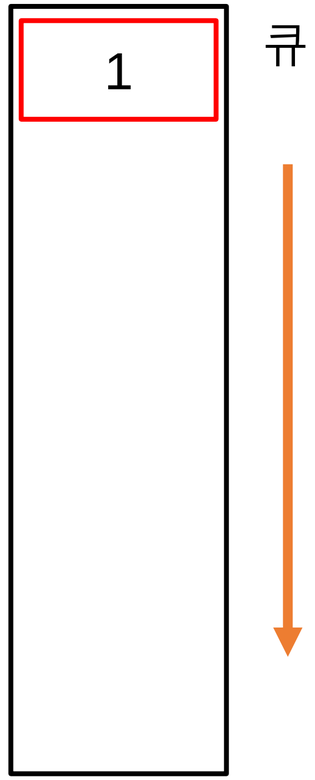
- 너비 우선 탐색의 구현에는 먼저 들어간 게 먼저 빠져나오는 자료구조인 큐를 사용한다
- 보통 queue stl이나 같은 기능을 할 수 있는 컨테이너를 사용해서 구현

그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

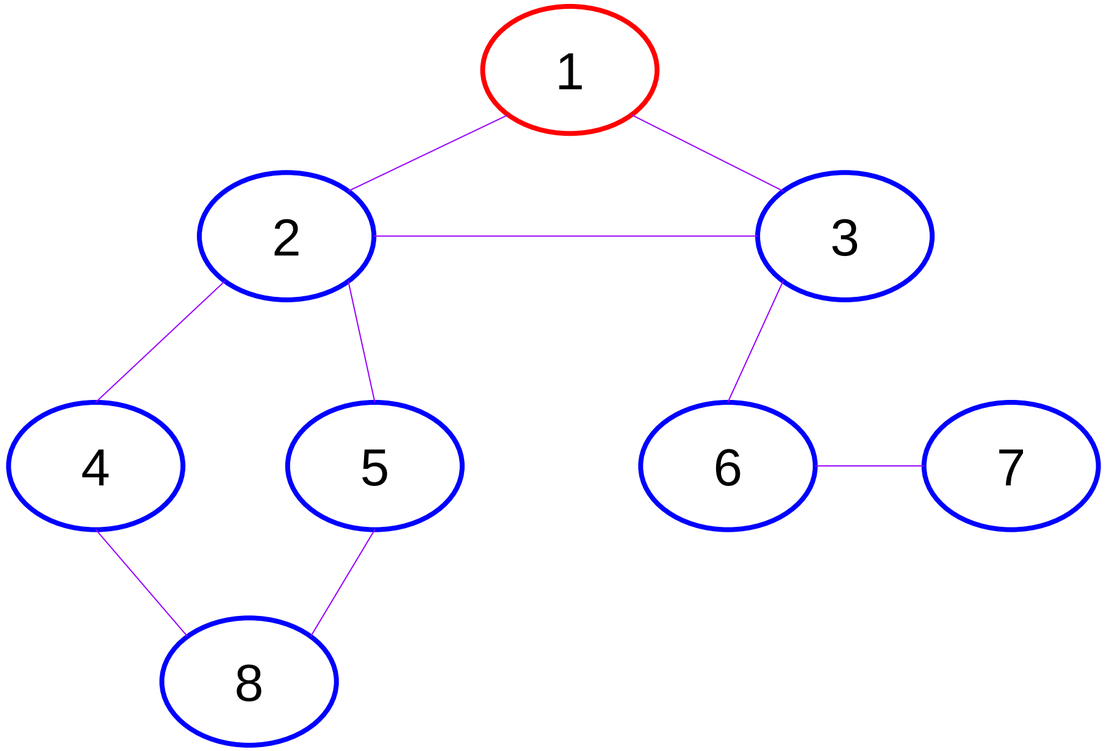


- 1번 정점에서 시작한다

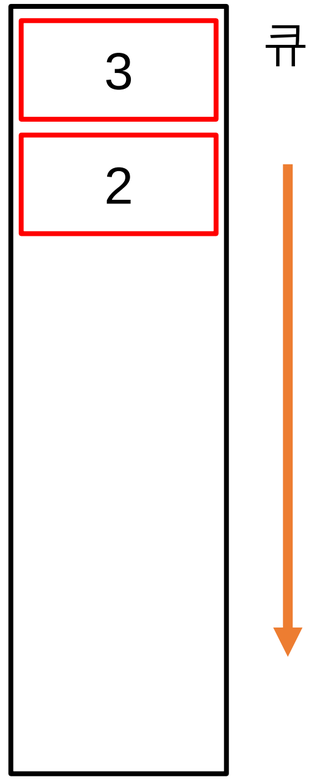


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

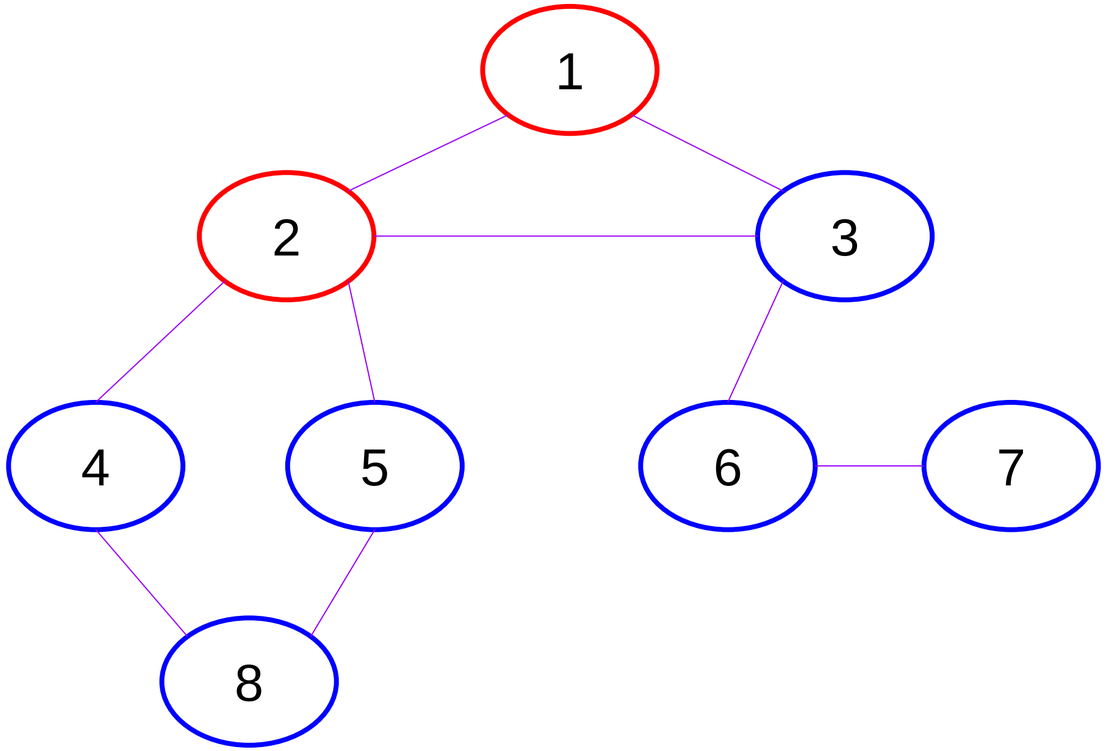


- 1번 정점을 큐에서 빼서 방문하고 아직 큐에 넣은 적이 없는 1번의 자식 정점들을 큐에 삽입한다

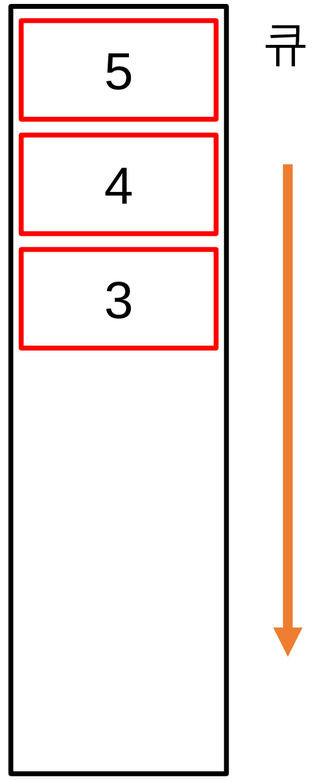


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

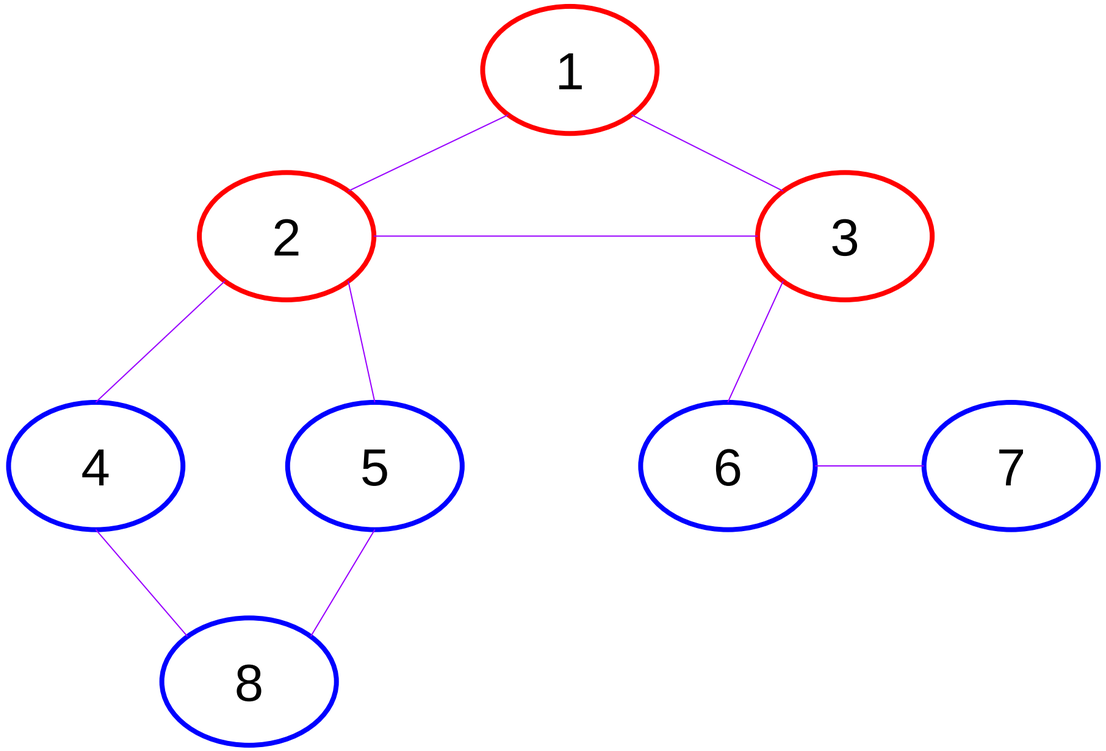


- 2번 정점을 큐에서 빼서 방문하고 아직 큐에 넣은 적이 없는 2번의 자식 정점들을 큐에 삽입한다

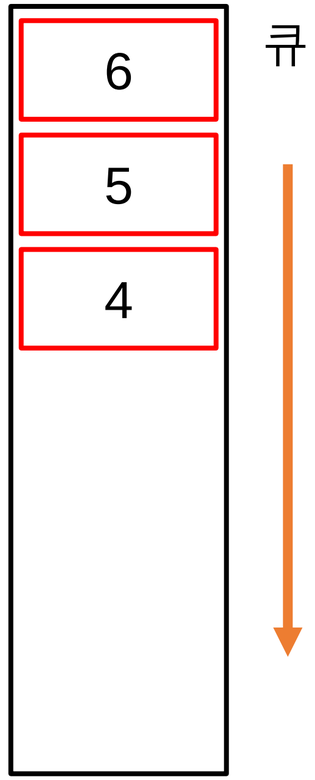


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

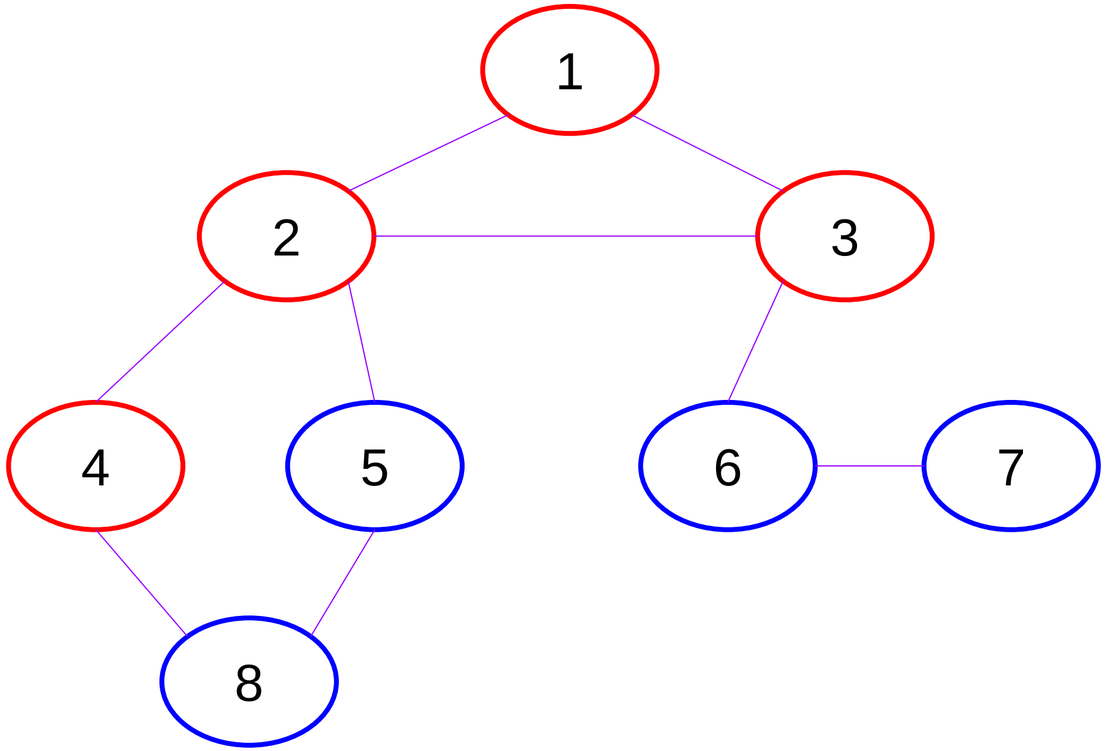


- 3번 정점을 큐에서 빼서 방문하고 아직 큐에 넣은 적이 없는 3번의 자식 정점들을 큐에 삽입한다

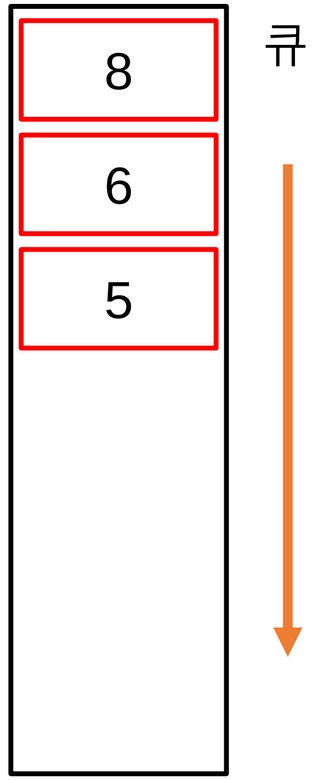


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

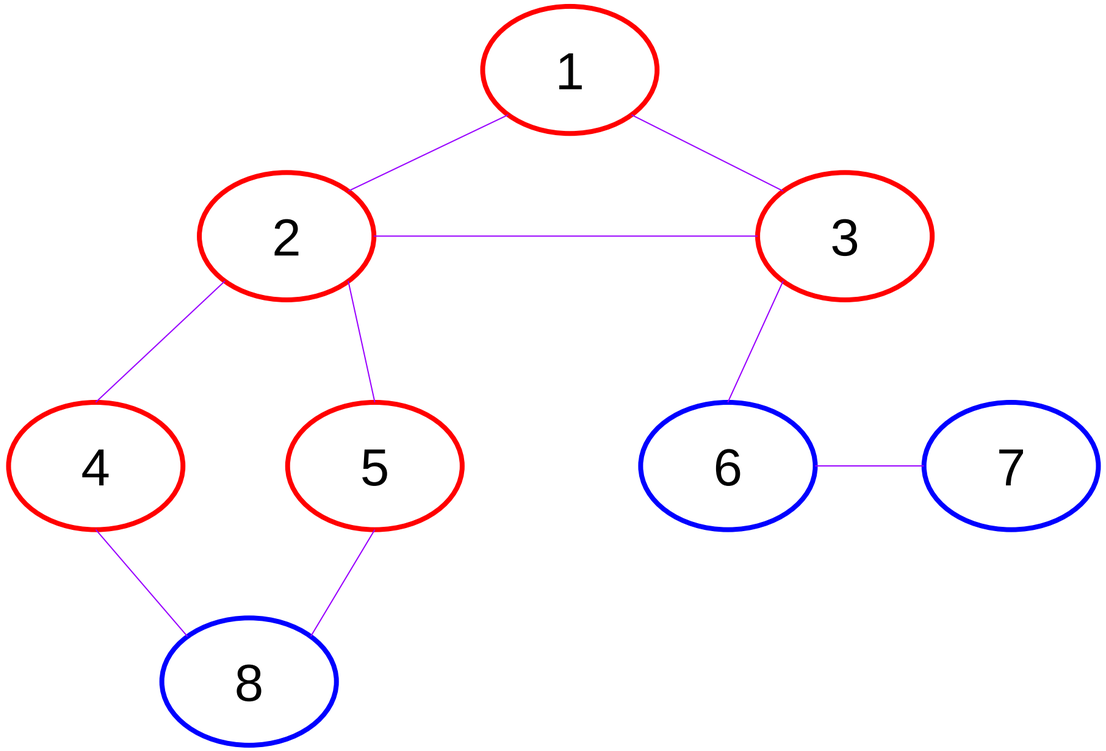


- 4번 정점을 큐에서 빼서 방문하고 아직 큐에 넣은 적이 없는 4번의 자식 정점들을 큐에 삽입한다

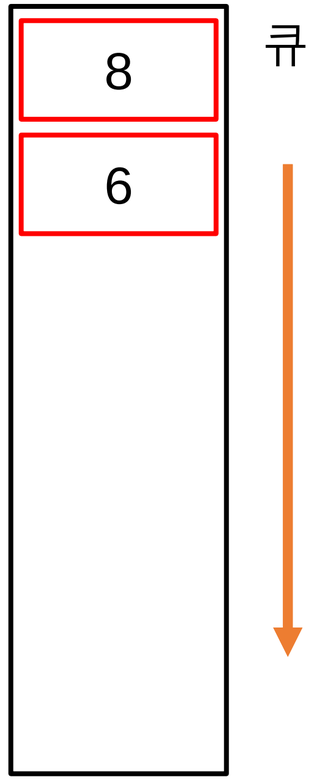


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

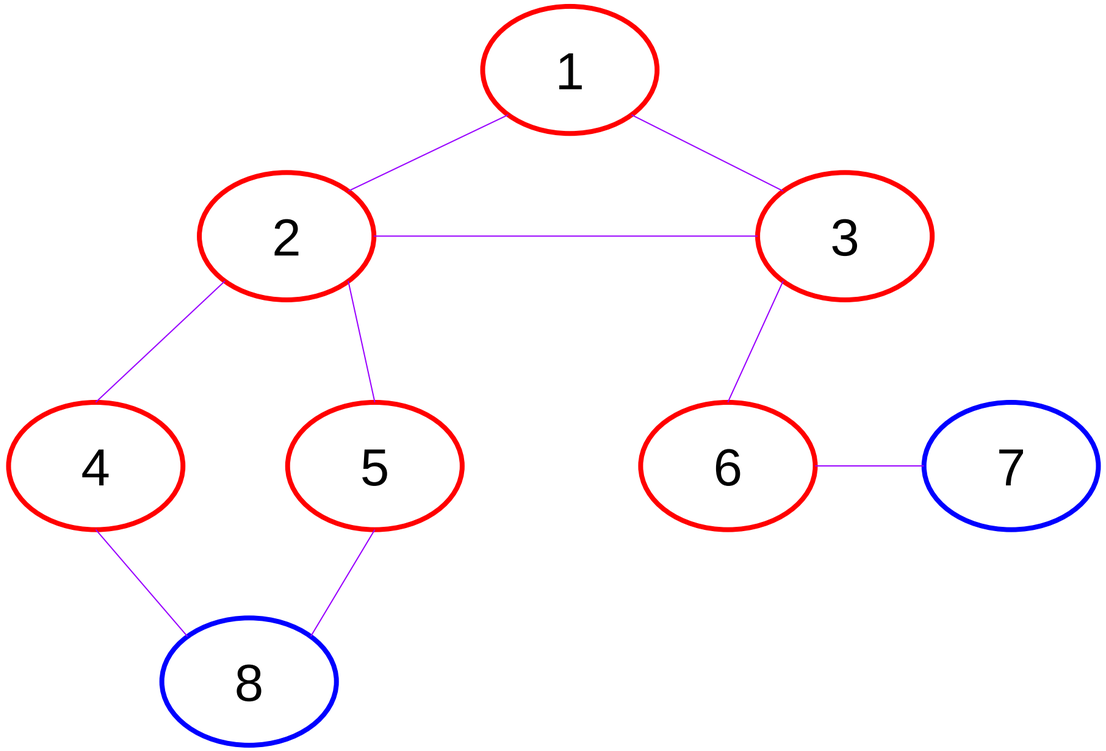


- 5번 정점을 큐에서 빼서 방문하고 아직 큐에 넣은 적이 없는 5번의 자식 정점들을 큐에 삽입한다. 그러나 그런 정점이 없으므로 그냥 넘어갈 것

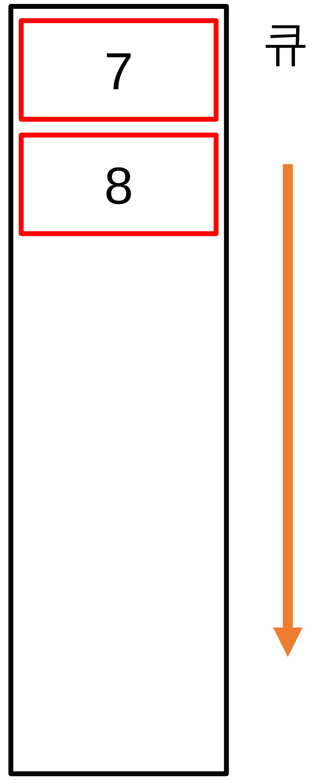


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

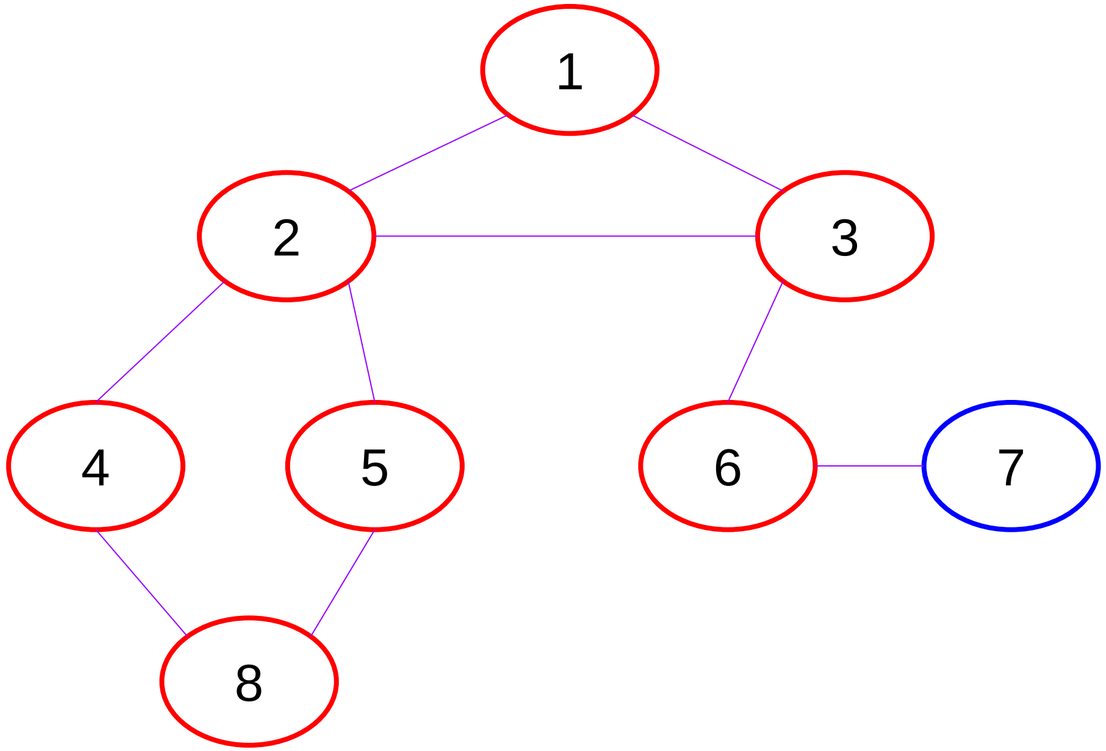


- 6번 정점을 큐에서 빼서 방문하고 아직 큐에 넣은 적이 없는 6번의 자식 정점들을 큐에 삽입한다

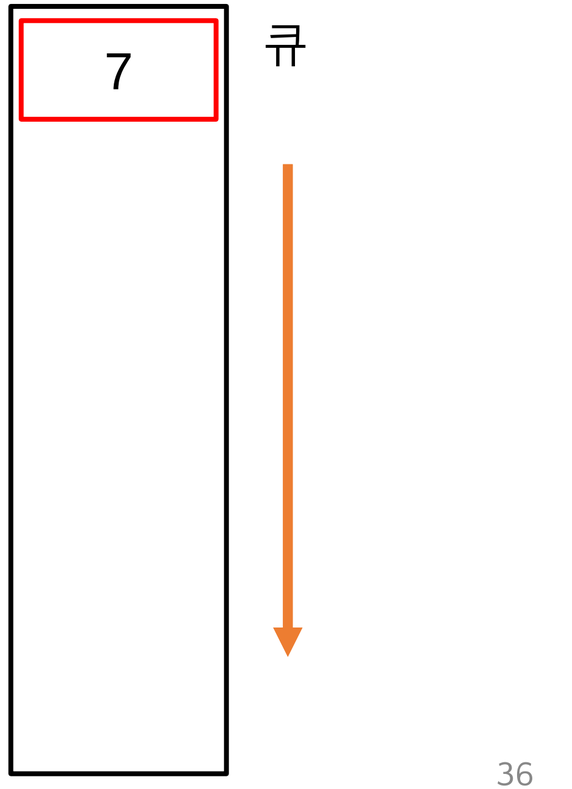


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

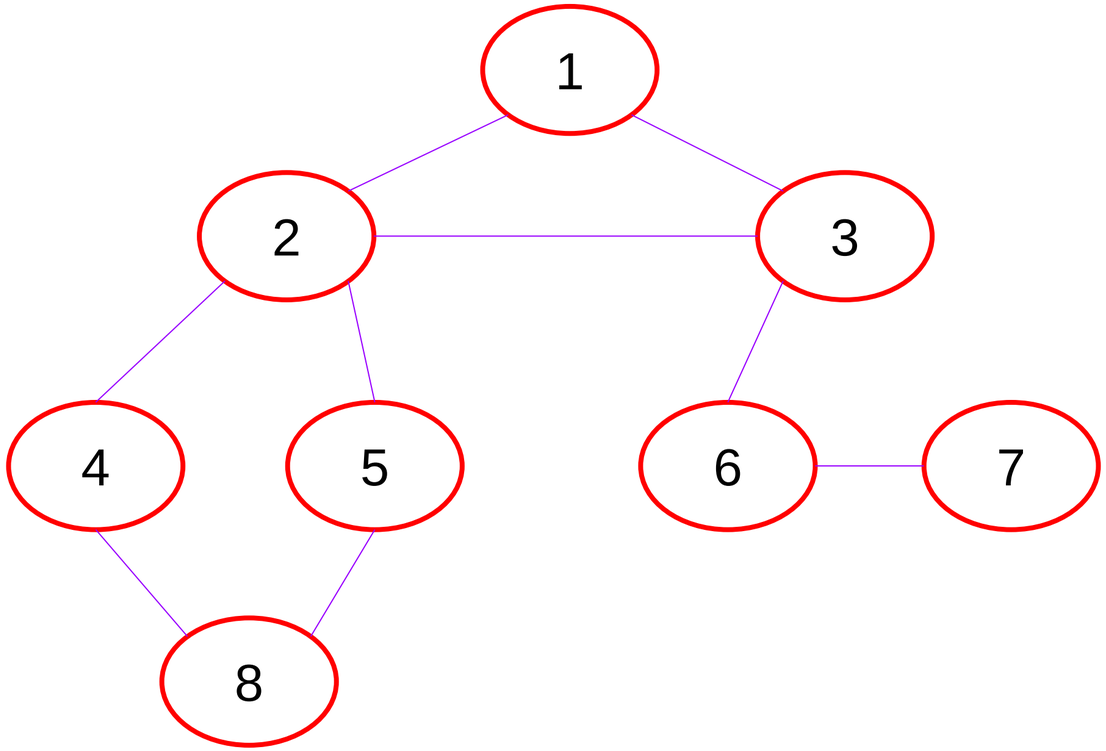


- 8번 정점을 큐에서 빼서 방문하고 아직 큐에 넣은 적이 없는 8번의 자식 정점들을 큐에 삽입한다. 그러나 그런 정점이 없으므로 그냥 넘어갈 것

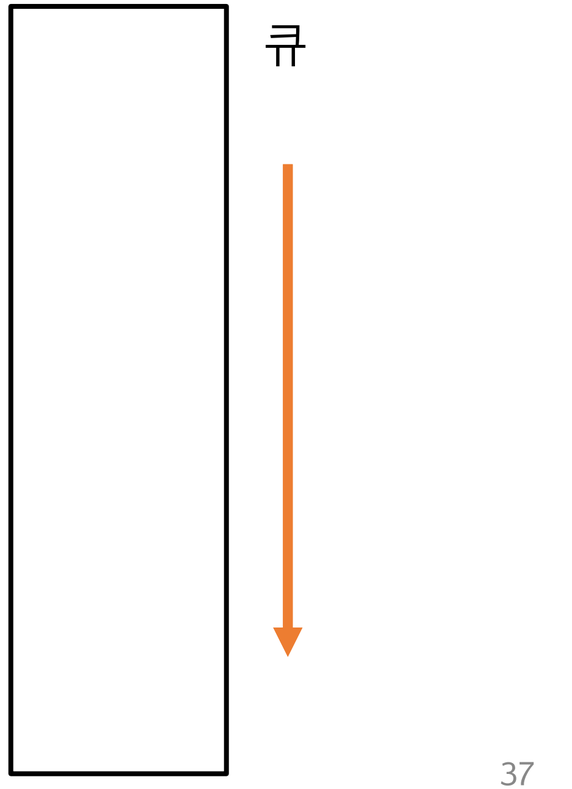


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**

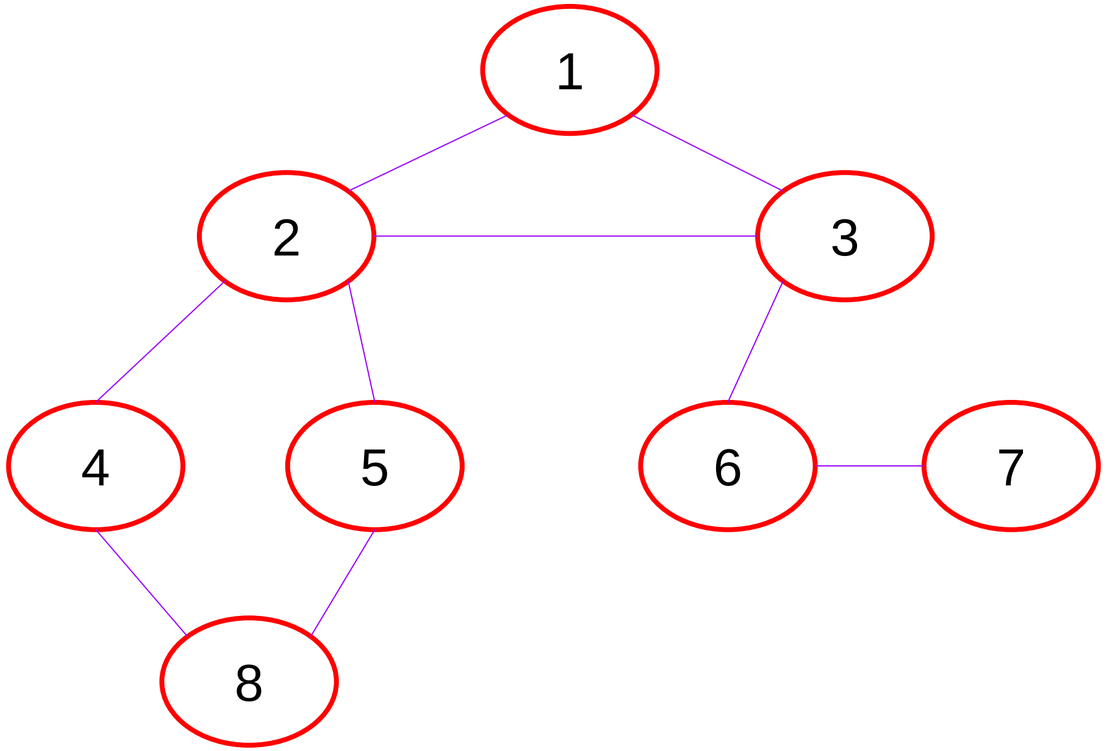


- 7번 정점을 큐에서 빼서 방문하고 아직 큐에 넣은 적이 없는 7번의 자식 정점들을 큐에 삽입한다. 그러나 그런 정점이 없으므로 그냥 넘어갈 것

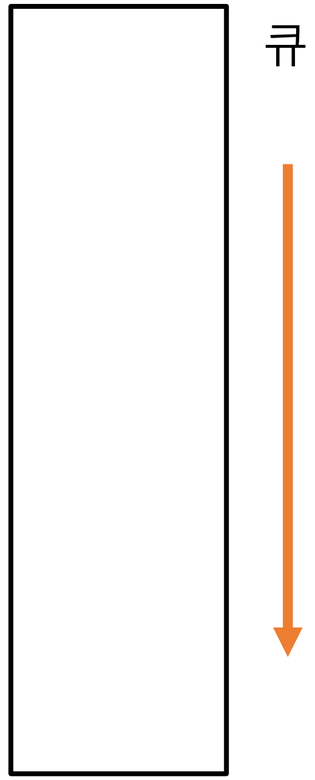


그래프 - 너비 우선 탐색

*** 그래프의 너비 우선 탐색**



- 그래프의 모든 정점을 방문하였고 큐는 비었다



그래프 - 너비 우선 탐색 : 구현

* 너비 우선 탐색 구현 - 큐

- 설명과 그림에서 보았듯이 기본적으로 큐를 활용한다
- 정점들이 큐에 들어가고 빠지는 과정을 추적해 보면 점차 너비를 넓혀 나가는 걸 알 수 있다

```
vector<int> adj[1005];
int visited[1005]={0};
int dist[1005]={0};

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

- 현재 방문하는 정점의 자식 정점 중 아직 방문하지 않은 정점을 큐에 넣는다
- 큐의 역할을 할 수 있는 다른 stl들도 있다. list, deque, 심지어 배열을 이용해 구현한 야매 스택으로도 가능하다.

그래프 - 너비 우선 탐색 : 구현

* 너비 우선 탐색 구현 - 큐

- 설명과 그림에서 보았듯이 기본적으로 큐를 활용한다
- 정점들이 큐에 들어가고 빠지는 과정을 추적해 보면 점차 너비를 넓혀 나가는 걸 알 수 있다

```
vector<int> adj[1005];
int visited[1005]={0};
int dist[1005]={0};

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

- 현재 방문하는 정점의 자식 정점 중 아직 방문하지 않은 정점을 큐에 넣는다
- 큐의 역할을 할 수 있는 다른 stl들도 있다. list, deque, 심지어 배열을 이용해 구현한 야매 스택으로도 가능하다.
- 앞에서 본 배열을 이용한 스택과 같이 큐도 양끝을 관리하는 인덱스만 적절하게 만들어 주면 배열로 대충 구현할 수 있다
- 하지만 중요한 것은 그 구조의 이해!

그래프 - 탐색 방법 두 가지

* 깊이 우선 탐색 vs 너비 우선 탐색

- 깊이 우선 탐색은 스택을, 너비 우선 탐색은 큐를 사용한다
- 깊이 우선 탐색은 사이클 검출이나, 재귀적 구조를 띠는 것을 이용하여 dp에 사용할 수 있음
- 일단 최대한 깊이 파고들어가므로 어떤 경로 하나를 찾고 나서 프로세스를 종료시키기도 쉬움
- 너비 우선 탐색은 최단거리를 찾을 수 있는 특성 때문에 그래프 모델링에 많이 쓰임
- 특정 상태 A에서 목표로 하는 상태 B까지 가기 위한 최소 연산 횟수 등을 구하는 데에 응용

그래프 - 탐색 방법 두 가지

* 깊이 우선 탐색 vs 너비 우선 탐색

- 깊이 우선 탐색은 스택을, 너비 우선 탐색은 큐를 사용한다
- 깊이 우선 탐색은 사이클 검출이나, 재귀적 구조를 띠는 것을 이용하여 dp에 사용할 수 있음
- 일단 최대한 깊이 파고들어가므로 어떤 경로 하나를 찾고 나서 프로세스를 종료시키기도 쉬움
- 너비 우선 탐색은 최단거리를 찾을 수 있는 특성 때문에 그래프 모델링에 많이 쓰임
- 특정 상태 A에서 목표로 하는 상태 B까지 가기 위한 최소 연산 횟수 등을 구하는 데에 응용
- 실제로 문제를 풀 때, 정점 하나하나가 어떻게 탐색되는지를 쫓아갈 수 있어야 하는 건 아님
- 중요한 건 스스로가 사용하는 그래프 탐색 방법이 어떤 흐름으로 이루어지는지를 이해하고, 그 흐름으로 생각할 수 있는 능력

2022 Winter Algorithm Camp

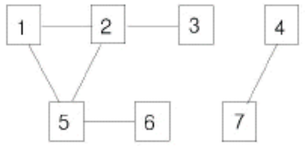
그래프 탐색 - 예시문제 : 2606. 바이러스

문제

선정 의도 : 간단한 그래프 탐색을 통해 풀 수 있는 문제를 풀어 보자

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 >

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.

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.

그래프 탐색 - 예시문제 : 2606. 바이러스

* 문제의 아이디어

- 1번 컴퓨터에서 간선을 통해 도달할 수 있는 컴퓨터의 수를 구하면 되는 문제이다
- 이렇게 서로가 간선을 통해 도달할 수 있는 경로가 존재하는 정점들을 묶어서 ‘연결 요소 (connected component)’ 라고 부르기도 한다(이 문제에서 크게 중요한 개념은 아님)

그래프 탐색 - 예시문제 : 2606. 바이러스

* 문제의 아이디어

- 1번 컴퓨터에서 간선을 통해 도달할 수 있는 컴퓨터의 수를 구하면 되는 문제이다
- 이렇게 서로가 간선을 통해 도달할 수 있는 경로가 존재하는 정점들을 묶어서 ‘연결 요소 (connected component)’ 라고 부르기도 한다(이 문제에서 크게 중요한 개념은 아님)
- 즉 1번 컴퓨터에서 시작해서 도달하는 정점의 개수를 세 주기만 하면 되는 문제
- 앞의 구현 코드에서 //do something 이 써 있는 부분에 답을 증가시켜 주는 코드를 추가하기만 하면 됨

2022 Winter Algorithm Camp

그래프 탐색 - 예시문제 : 10026. 적록색약

선정 의도 : 1. 연결 요소의 개수를 세어 보자
2. 2차원 상에서 그래프 탐색을 어떻게 진행하는지 익히자

문제

적록색약은 빨간색과 초록색의 차이를 거의 느끼지 못한다. 따라서, 적록색약인 사람이 보는 그림은 아닌 사람이 보는 그림과는 좀 다를 수 있다.

크기가 N×N인 그리드의 각 칸에 R(빨강), G(초록), B(파랑) 중 하나를 색칠한 그림이 있다. 그림은 몇 개의 구역으로 나뉘어져 있는데, 구역은 같은 색으로 이루어져 있다. 또, 같은 색상이 상하좌우로 인접해 있는 경우에 두 글자는 같은 구역에 속한다. (색상의 차이를 거의 느끼지 못하는 경우도 같은 색상이라 한다)

예를 들어, 그림이 아래와 같은 경우에

```
RRRBB
GGBBB
BBBRR
BBRRR
RRRRR
```

적록색약이 아닌 사람이 봤을 때 구역의 수는 총 4개이다. (빨강 2, 파랑 1, 초록 1) 하지만, 적록색약인 사람은 구역을 3개 볼 수 있다. (빨강-초록 2, 파랑 1)

그림이 입력으로 주어졌을 때, 적록색약인 사람이 봤을 때와 아닌 사람이 봤을 때 구역의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N이 주어진다. ($1 \le N \le 100$)

둘째 줄부터 N개 줄에는 그림이 주어진다.

그래프 탐색 - 예시문제 : 10026. 적록색약

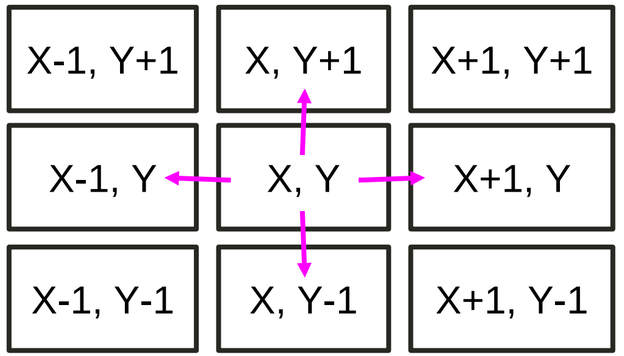
* 문제의 아이디어

- 처음 문제가 주어진 상태에서의 영역의 개수를 한 번 세고, 빨간색과 초록색을 같게 봤을 때의 영역 개수를 한 번 센다
- 어떻게 영역 개수를 셀 것인가?
- 같은 영역에 속한 색상은 상, 하, 좌, 우 중 한 방향으로 인접해 있다
- 따라서 각 격자를 하나의 정점으로 보고 각 정점은 상, 하, 좌, 우에 있는 격자(정점)와 인접해 있는 걸로 생각하면 주어진 그리드를 그래프로 생각할 수 있다
- 그런 그래프에서 연결 요소를 세면 된다

그래프 탐색 - 예시문제 : 10026. 적록색약

*** 2차원 평면에서의 그래프 탐색**

- 처음 보면 낯설 수 있지만 매우 유명한 아이디어
- 인접 리스트로 그래프를 나타내는 대신 다른 방식을 사용한다
- 모든 정점에 대해, 그 정점과 인접한 정점은 상, 하, 좌, 우 방향에만 있다
- 따라서 각 정점의 좌표 (x,y) 에 대해 (x+1, y), (x-1, y), (x, y+1), (x, y-1) 중 특정 조건을 만족하는 것과만 인접해 있는 것



그래프 탐색 - 예시문제 : 10026. 적록색약

* 2차원 평면에서의 그래프 탐색

- 각 정점의 좌표 (x,y) 에 대해 (x+1, y), (x-1, y), (x, y+1), (x, y-1) 중 특정 조건을 만족하는 것과만 인접해 있는 것
- 각 정점의 인접 리스트에서 정점을 빼서 확인 후 큐에 넣는 것이 아니라, 즉각적으로 그 정점의 상, 하, 좌, 우 정점을 생성해 주는 방식이다
- 경계에 대해서는 적절히 처리해 준다

```
int dx[4] = { 1,-1,0,0 };
int dy[4] = { 0,0,1,-1 };

int n, ans = 0, ans2 = 0;
char arr[105][105];
int visited[105][105] = { 0 };
```

```
while (!q.empty()) {
    coor cur = q.back(); q.pop_back();
    int x = cur.x, y = cur.y;
    for (int k = 0; k < 4; k++) {
        int nx = x + dx[k], ny = y + dy[k];
        if (visited[nx][ny])continue;
        if (nx < 0 || ny < 0 || nx >= n || ny >= n)continue;
        if (arr[nx][ny] != arr[x][y])continue;
        visited[nx][ny] = 1;
        q.push_front({ nx,ny });
    }
}
```

상,하,좌,우 정점의 생성

그래프 탐색 - 연결 요소 문제

*** 필수 문제**

- 2583. 영역 구하기
- 갈 수 없는 영역에 대한 관리가 필요한 문제다. 갈 수 없는 영역을 미리 다 색칠해 놓는다는 아이디어

*** 연습 문제**

- 2667. 단지번호붙이기
- 11724. 연결 요소의 개수
- 1325. 효율적인 해킹
- 1926. 그림

너비 우선 탐색 - 최단거리

* 너비 우선 탐색의 특징

- 앞에서 보았듯이, 시작 정점에서 점점 한 단계씩 퍼져나감
- 만약 어떤 정점이 시작 정점으로부터 깊이 2에 있다면, 그 정점은 무조건 깊이 1인 정점을 다 방문하고 나서 방문될 것이다
- 이러한 특성을 이용해서 최단거리를 구하는 데에 이용될 수 있음
- 거리 배열을 두고 dist[next]=dist[cur]+1 과 같은 방식으로 구현
- next 정점은 cur 정점에 비해서 시작 정점으로부터의 거리가 1 긴 것이 당연하다

```
vector<int> adj[1005];
int visited[1005]={0};
int dist[1005]={0};

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

너비 우선 탐색 - 최단거리

* 너비 우선 탐색의 특징

- 앞에서 보았듯이, 시작 정점에서 점점 한 단계씩 퍼져나감
- 만약 어떤 정점이 시작 정점으로부터 깊이 2에 있다면, 그 정점은 무조건 깊이 1인 정점을 다 방문하고 나서 방문될 것이다
- 이러한 특성을 이용해서 최단거리를 구하는 데에 이용될 수 있음
- 거리 배열을 두고 dist[next]=dist[cur]+1 과 같은 방식으로 구현
- next 정점은 cur 정점에 비해서 시작 정점으로부터의 거리가 1 긴 것이 당연하다
- 특정 상태에서 어떤 상태가 되는 데에 몇 번의 연산이 필요한가? 를 구하는 문제에서 자주 응용됨(예시 문제 나올 것)

```cpp
vector<int> adj[1005];
int visited[1005]={0};
int dist[1005]={0};

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

2022 Winter Algorithm Camp

그래프 탐색 - 예시문제 : 14940. 쉬운 최단거리

선정 의도 : 1. 2차원 상의 그래프 탐색을 또 한번 해보자
2. BFS로 최단거리를 구해보자

문제

지도가 주어지면 모든 지점에 대해서 목표지점까지의 거리를 구하여라.

문제를 쉽게 만들기 위해 오직 가로와 세로로만 움직일 수 있다고 하자.

입력

지도의 크기 n과 m이 주어진다. n은 세로의 크기, m은 가로의 크기다.($2 \le n \le 1000$, $2 \le m \le 1000$)

다음 n개의 줄에 m개의 숫자가 주어진다. 0은 갈 수 없는 땅이고 1은 갈 수 있는 땅, 2는 목표지점이다. 입력에서 2는 단 한개이다.

출력

각 지점에서 목표지점까지의 거리를 출력한다. 원래 갈 수 없는 땅인 위치는 0을 출력하고, 원래 갈 수 있는 땅인 부분 중에서 도달할 수 없는 위치는 -1을 출력한다.

그래프 탐색 - 예시문제 : 14940. 쉬운 최단거리

*** 문제의 아이디어**

- 위에서 본 대로 2차원에서의 너비 우선 탐색을 하면 된다
- 시작점은 당연히 목표 지점
- 거리 배열을 사용하여 시작점에서 각 정점까지의 최단거리를 구하는 코드를 삽입할 수 있다
- 이때 정점이 (x,y) 좌표로 표시되므로 정점은 구조체나 pair로 관리한다
- 따라서 거리 배열이나 방문 관리 배열도 2차원으로 선언한다(1차원으로 관리하는 방법도 있다)
- 이 문제에서는 갈 수 없는 지점이 있다는 것에 주의하라

그래프 탐색 - 예시문제 : 14940. 쉬운 최단거리

* 문제의 아이디어

- 위에서 본 대로 2차원에서의 너비 우선 탐색을 하면 된다
- 시작점은 당연히 목표 지점
- 거리 배열을 사용하여 시작점에서 각 정점까지의 최단거리를 구하는 코드를 삽입할 수 있다
- 이때 정점이 (x,y) 좌표로 표시되므로 정점은 구조체나 pair로 관리한다
- 따라서 거리 배열이나 방문 관리 배열도 2차원으로 선언한다(1차원으로 관리하는 방법도 있다)
- 이 문제에서는 갈 수 없는 지점이 있다는 것에 주의하라
- 갈 수 없는 지점을 관리하는 방법은 몇 가지 있다
- 대표적으로는 큐에 넣기 전에 갈 수 없는 지점인지 확인하는 방식

그래프 탐색 - 최단거리 문제

* 연습 문제

- 좀 생각을 해서 최단거리를 찾아야 하는 문제들로 꼽아 보았다
- 17836. 공주님을 구해라!
- 7576. 토마토
- 1697. 숨바꼭질

2022 Winter Algorithm Camp

그래프 모델링

* 정리

- 그래프의 개념과 그래프 탐색 방법 2가지에 관해 알아보았다
- 정점과 정점들을 연결하는 간선으로 이루어진 그래프
- 그리고 스택을 사용해서 그래프의 모든 정점을 탐색하는 깊이 우선 탐색
- 큐를 사용해서 그래프의 모든 정점을 탐색하는 너비 우선 탐색

* 그래프의 응용 한 스푼

- 그래프의 기초를 설명하기 위해서 이미 많은 슬라이드를 써 버렸다
- 하지만 강의로 다 다룰 수 없을 정도로 많은 그래프의 응용과 발상들이 있음
- 그 중 다른 곳에도 많이 쓰이는 아이디어 하나만 배워보도록 하자

그래프 모델링

* 상황에 대한 모델링

- ‘이 문제를 풀기 위해서 어떤 자료구조를 어떠어떠한 방법으로 쓰라’고 대놓고 지시해 주는 문제는 거의 없다
- 결국 우리가 알고리즘 문제를 푼다는 것은 문제에 주어진 상황을 우리가 아는 자료구조와 알고리즘을 사용해서 풀 수 있도록 모델링하는 것
- 예를 들어 아까의 ‘바이러스’문제는 네트워크를 그래프로 치환하고 바이러스의 전파를 그래프 탐색으로 모델링한 문제
- 우리가 지금까지 푼 예제들도 문제의 상황을 BFS, DFS로 모델링한 것
- 중요한 것은 문제의 상황을 풀어내는 것이므로 누군가는 또다른 방법으로 풀 수도 있다

그래프 모델링

* 상황에 대한 모델링

- 그 중 그래프로 모델링해서 푸는 거라고 의심할 수 있는 문제 상황들이 있다
- 대표적으로는 A상태에서 B상태로 가는 데에 거쳐야 하는 상태나 연산들이 있고, B상태까지 가기 위해 필요한 최소 연산 횟수를 구하는 문제
- 각 상태가, 그 상태에 연산을 가해서 나올 수 있는 새로운 상태들과 연결되어 있다고 하고 A점을 시작점으로 BFS를 시행하여 B 상태까지의 최단거리를 구한다
- 문제 하나로 이해해 보자

2022 Winter Algorithm Camp

그래프 모델링 예시문제 : 16953. A → B

문제

선정 의도 : 1. 간단한 그래프 모델링을 해보자

정수 A를 B로 바꾸려고 한다. 가능한 연산은 다음과 같은 두 가지이다.

- 2를 곱한다.
- 1을 수의 가장 오른쪽에 추가한다.

A를 B로 바꾸는데 필요한 연산의 최솟값을 구해보자.

입력

첫째 줄에 A, B ($1 \leq A < B \leq 10^9$)가 주어진다.

출력

A를 B로 바꾸는데 필요한 연산의 최솟값에 1을 더한 값을 출력한다. 만들 수 없는 경우에는 -1을 출력한다.

그래프 모델링 예시문제 : 16953. A → B

*** 문제의 아이디어**

- 정수 A에서 시작해서 B로 가는 최단거리를 구해야 한다
- 아까 배운 BFS로 최단거리 구하기를 응용할 수 있다
- 각 정수가 정점이라 한다면, 간선은 어떻게 연결되어 있는가?
- 임의의 정수 X는 2*X와 10*X+1, 두 정점과 연결되어 있다고 간주할 수 있다
- 현재 방문하고 있는 정점 X에 대해 2*X와 10*X+1 을 큐에 넣는 방식으로 진행한다

그래프 모델링 예시문제 : 16953. A → B

* 문제의 아이디어

- 정수 A에서 시작해서 B로 가는 최단거리를 구해야 한다
- 아까 배운 BFS로 최단거리 구하기를 응용할 수 있다
- 각 정수가 정점이라 한다면, 간선은 어떻게 연결되어 있는가?
- 임의의 정수 X는 2*X와 10*X+1, 두 정점과 연결되어 있다고 간주할 수 있다
- 현재 방문하고 있는 정점 X에 대해 2*X와 10*X+1 을 큐에 넣는 방식으로 진행한다
- 주의 1: 정점을 나타내는 X값이 커질 수 있으므로 거리 배열 대신 map 등으로 거리를 관리
- 주의 2: 정점에 연산을 무한히 가할 수 있으므로 B 정점에 도달하면 BFS를 종료시켜 줘야 함

```
if(cur==b){ ans=dist[cur]; break;}
```

2022 Winter Algorithm Camp

그래프 모델링 - 연습문제

* 연습문제

- 23085. 판치기

  뒷면이 나와 있는 동전의 개수가 같은 상태는 모두 같은 상태라는 점을 이용해 문제 상황을 그래프로 모델링

- (Challenging) 1963. 소수 경로

  한 번에 한 자리를 바꿨을 때 얻을 수 있는 모든 네 자리 소수들과 연결되어 있다고 생각할 수 있다

2022 Winter Algorithm Camp

그래프 탐색의 응용

*** 많은 응용들**

- 사이클 검출, 트리 dp, 탑다운 dp, 백트래킹, 이분 그래프 등 DFS, BFS의 수많은 응용이 있다
- 그 외에도 다음 시간에 다룰 트리부터 최단경로 알고리즘, 위상 정렬, SCC, 네트워크 플로우 등 수도 없는 알고리즘들이 그래프 위에서 전개됨
- 그래프가 무엇인지 잘 알아 놓고, 문제를 풀면서 그래프를 이용하는 방법을 익혀 놓자
- 특히 DFS를 이용한 사이클 검출 같은 경우 난이도도 그렇게 높지 않고 배울 만 하다
- 강의에서도 다루려고 하다가 시간 관계상 삭제당함

그래프 탐색의 응용

* 다른 알고리즘과의 결합

- 대회에서 문제를 풀어내는 것은 여러 알고리즘들을 복합하여 써야 할 때가 많음
- 어떤 제한을 가지고 A지점에서 B지점으로 가는 게 가능한지를 이분탐색으로 판단해야 하는 경우 (13905. 세부, 1939. 중량제한)
- BFS하면서 어떤 처리를 해 주거나 안 해 주는 것을 DP로 처리해야 하는 경우(2206. 벽 부수고 이동하기)
- 보통 알고리즘 하나를 떠올리는 것은 상대적으로 쉽다. 그러나 특정 알고리즘을 이용한 최적화가 필요한 때에 이런 경우가 많이 보인다.

그래프 탐색의 응용

* 다른 알고리즘과의 결합

- 대회에서 문제를 풀어내는 것은 여러 알고리즘들을 복합하여 써야 할 때가 많음
- 어떤 제한을 가지고 A지점에서 B지점으로 가는 게 가능한지를 이분탐색으로 판단해야 하는 경우 (13905. 세부, 1939. 중량제한)
- BFS하면서 어떤 처리를 해 주거나 안 해 주는 것을 DP로 처리해야 하는 경우(2206. 벽 부수고 이동하기)
- 보통 알고리즘 하나를 떠올리는 것은 상대적으로 쉽다. 그러나 특정 알고리즘을 이용한 최적화가 필요한 때에 이런 경우가 많이 보인다.
- 그래프는 문제의 상황을 나타내기 위해서 쓰이는 도구일 뿐이다. 그 상황 위에서 무엇을 해야 하는지를 고민할 것