그래프 표현 방법

그래프는 **정점(Vertex)**과 **정점들을 연결하는 간선(Edge)**으로 이루어져 있다. 여기서 정점은 어떤 자료 등을 표현하기 위해 사용하는 수단이며 간선은 정점들 즉, 자료 간의 관계를 나타내기 위해 사용한다.

그래프의 표현 방법에 대해 알기에 앞서 그래프와 관련된 용어들을 알아보자?

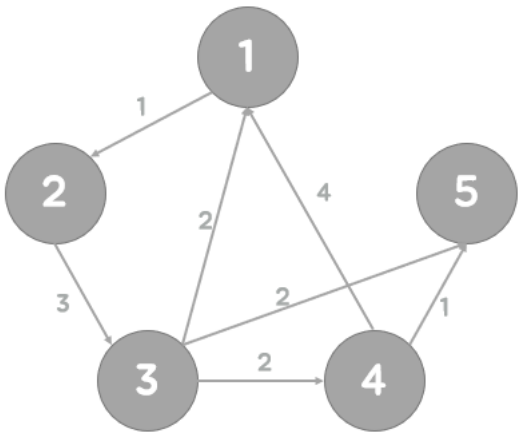
- **정점**
  : **자료를 표현**하기 위해 사용하며 노드(node)라고도 표현한다.
- **간선**
  : 정점(자료)을 연결하며 **정점 사이의 관계를 표현**한다.
- **가중치(weight)**
  : **한 정점에서 다른 정점으로 이동하는 데 필요한 거리, 비용 등을 정의한 것**으로 간선에 할당된다. 가중치가 없는 그래프의 경우 가중치를 1 로 생각하면 좋다.
- **차수(degree)**
  : **그래프와 연결된 간선의 개수**를 뜻한다.
    - 입력 차수: 그래프가 방향 그래프일 경우, 해당 정점으로 들어오는 간선의 개수를 말한다.
    - 출력 차수: 그래프가 방향 그래프일 경우, 해당 정점에서 나가는 간선의 개수를 말한다.
- **단순 경로(simple path)**
  : 특정 출발점에서 도착점으로 이동하는 동안 **하나의 정점을 최대 한 번씩만 방문한 경로**를 말한다. 즉 한 정점은 두 번 이상 방문할 수 없음을 뜻한다.
- **사이클(cycle)**
  : **단순 경로의 출발점과 도착점이 같은 경우**를 말한다.
- **연결 요소(connected component)**
  : 그래프의 정점은 연결되지 않은 상태일 수 있다. 연결 요소는 연결 요소에 속한 모든 정점을 연결하는 경로가 있어야 하며, 서로 다른 연결 요소에 속한 정점끼리 연결하는 경로는 존재해선 안 된다.

그래프에는 용어도 많지만, 그 종류 또한 다양한다.

- **양방향(무방향) 그래프(undirected graph)**
  : **간선에 방향이 없는 그래프**로, 간선이 연결되어 있다면 해당 간선을 통해 두 정점 u, v 를 이동할 수 있다. 즉 u → v, v → u 둘은 같은 간선이다.
- **방향 그래프(directed graph)**
  : **간선에 방향이 있는 그래프**로 u → v, v → u 둘은 다른 간선이다.
- **완전 그래프(complete graph)**
  : **각 정점에서 자신을 제외한 모든 정점과 연결된 그래프**를 말한다. 이때 간선의 개수는 방향 그래프는 V(V - 1), 양방향 그래프는 $\frac{V(V-1)}{2}$ 이다.
- **다중 그래프(multigraph)**
  : **두 정점 사이에 간선이 여러 개 있는 그래프**를 말한다.
- **사이클 없는 방향 그래프(directed acyclic graph)**
  : **사이클이 없는 방향 그래프**를 말하며 줄여서 DAG 라고 표현하기도 한다.
- **부분 그래프(subgraph)**
  : **어떤 그래프에서 정점(들)이나 간선(들)을 제외하여 만든 그래프**를 말한다.
- **이분 그래프(bipartite graph)**
  : **인접한 정점끼리 서로 다른 색으로 칠했을 때, 모든 정점을 두 가지 색으로만 칠할 수 있는 그래프**를 말한다. 약간 다르게 표현하자면 두 개의 정점을 두 그룹으로 나누어 같은 그룹에 속한 정점끼리는 연결되어 있지 않은 그래프를 말한다.

그래프의 종류까지 알아보았다면 다음은 컴퓨터에서 그래프를 표현하는 방법이다. 컴퓨터에서 그래프를 표현하기 위해서는 **인접 행렬과 인접 리스트**, 두 가지 방법을 이용한다. 먼저 인접 행렬의 경우 행렬을 이용하여 정점 사이의 관계를 나타내는 방법이다.

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	3	0	0
3	2	0	0	2	2
4	4	0	0	0	1
5	0	0	0	0	0



위 그림은 방향 그래프를 인접 행렬을 이용해 표현한 것이다. 행렬의 특성상 특정 그래프를 표현하려면 정점의 제곱인 $O(V^2)$만큼의 공간을 필요로 하므로, 정점의 개수가 많아지는 경우에는 사용하기 어렵습니다. 일반적으로 그래프를 이용한 문제를 풀이하기 위해 그래프 내 두 정점이 연결되어 있는지를 판단하는 연산 또는 해당 정점에 연결된 모든 정점을 찾는 연산을 주로 이용한다. 인접 행렬의 경우, 두 정점이 연결되어 있는지 알고 싶다면 행렬의 i 행 j 열로 간단히 확인할 수 있어 $O(1)$ 시간 밖에 걸리지 않고, 특정 정점에 연결된 모든 정점을 찾기 위해 모든 정점의 개수만큼 일일이 확인해야 하므로 $O(V)$ 시간 안에 확인할 수 있다.

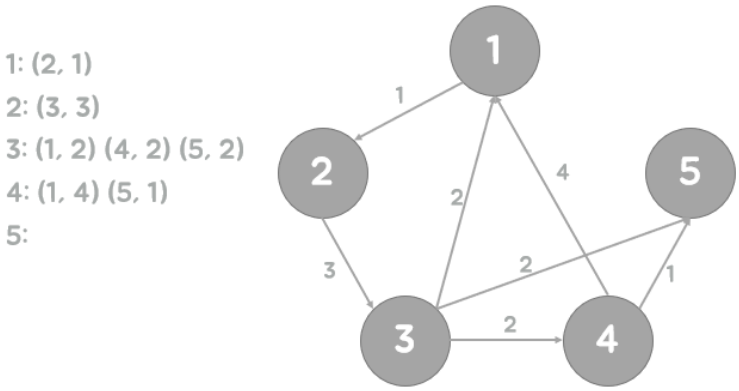
아래의 코드는 swift 로 인접 행렬을 구현하여 입력을 받는 예시이다.

```swift
let maxVertex = 1010 // 정점의 최대 개수
var adj = [[Int]](repeating: [Int](repeating: 0, count: maxVertex), count: maxVertex) // 인접 행렬로 사용할 배열 선언

var v = Int(readLine()!)! // 정점의 개수를 입력받음

for i in 0..<v {
    for j in 0..<v {
        let w = Int(readLine()!)! // 인접 행렬의 모든 요소를 입력받음
        adj[i][j] = w
    }
}
```

다음으로, 인접 리스트는 연결 리스트(Linked List) 또는 동적 배열인 벡터를 이용하여 정점에 연결된 모든 간선을 저장한다. 일반적으로 동적 배열(벡터)을 주로 이용한다.



위 그림은 인접 행렬을 이용하여 표현한 그래프와 동일한 그래프를 인접 리스트로 표현한 것이다. 1 번 정점에는 2 번 정점이 가중치 1 로 연결되어 있고, 2 번 정점에는 3 번 정점이 가중치 3 으로, 3 번 정점에는 1 번, 4 번, 5 번 정점이 각각 가중치 2, 2, 2 로 연결되어 있고, 4 번 정점에는 1 번, 5 번 정점이 각각 가중치 4, 1 로, 5 번 정점에는 어떠한 정점도 연결되어 있지 않다는 것을 뜻한다. 인접 리스트는 두 정점의 연결 여부를 확인하기 위해 해당 노드와 연결된 모든 요소를 확인하므로, 해당 정점의 차수인 $O(d)$의 시간이 걸립니다. 그뿐만 아니라 해당 정점과 연결된 모든 정점을 찾는 것 역시 $O(d)$의 시간이 걸립니다.