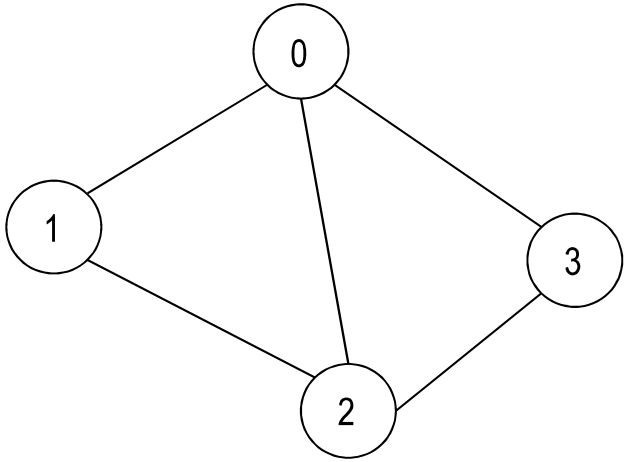
그래프

그래프

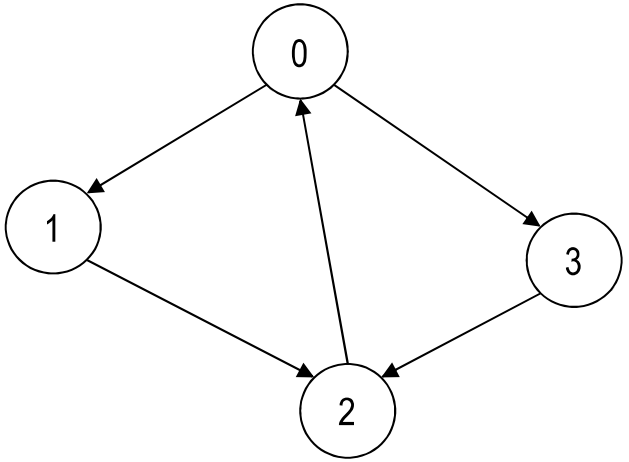
- 그래프(graph)는 정점(vertex)들의 집합 V와 간선(edge)들의 집합 E로 정의된다.
- 그래프는 간선의 방향성 유무에 따라 무방향 그래프와 방향 그래프로 구분된다.

V = { 0, 1, 2, 3 }
E = { (0,1), (0,2), (0,3), (1,2), (2,3) }



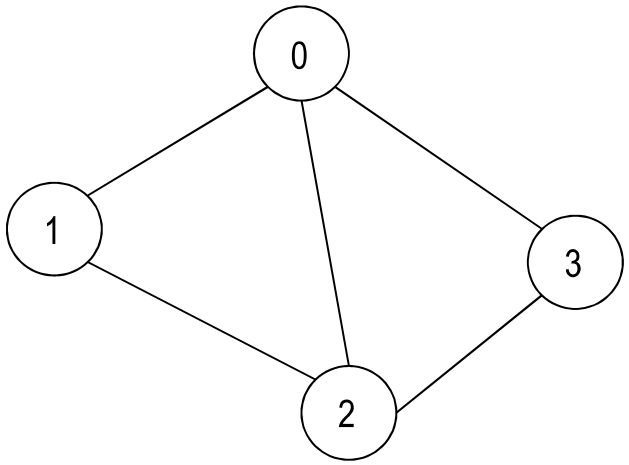
무방향 그래프(undirected graph)

V = { 0, 1, 2, 3 }
E = { <0,1>, <0,3>, <1,2>, <2,0>, <3,2>}

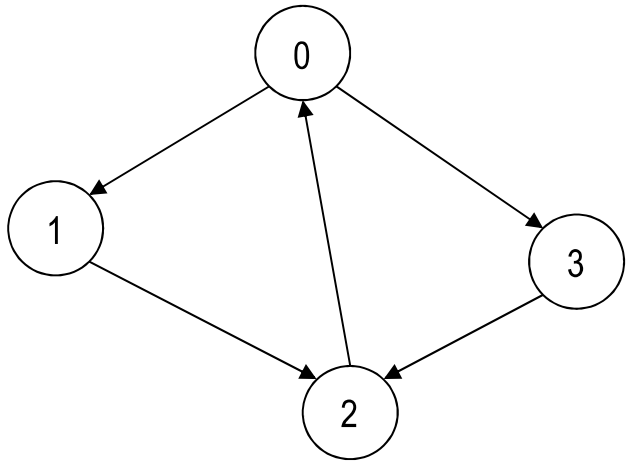


방향 그래프(directed graph)

그래프 표현: 인접행렬(adjacency matrix)

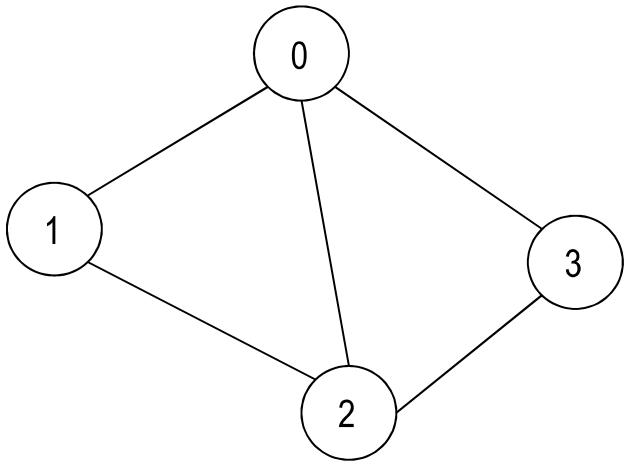


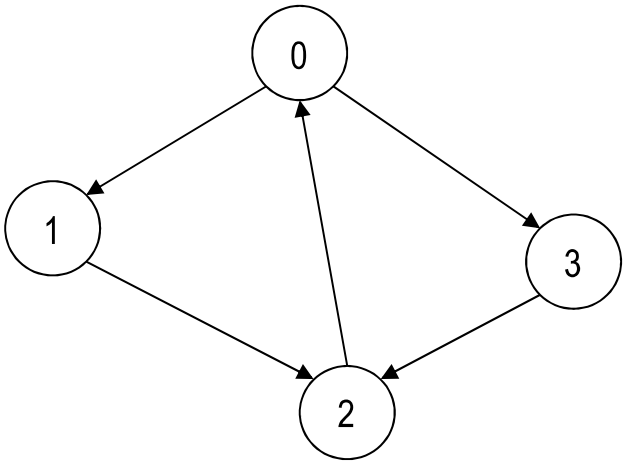
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	1
3	1	0	1	0

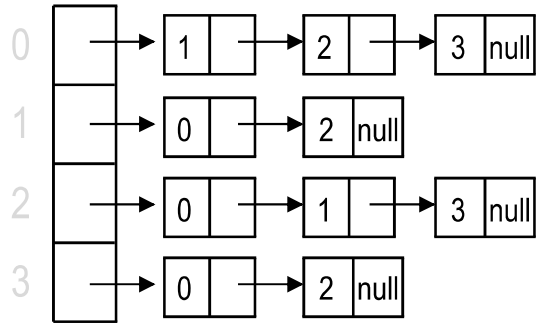


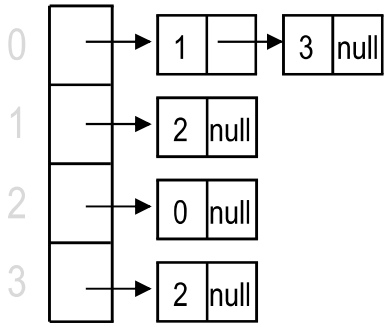
	0	1	2	3
0	0	1	0	1
1	0	0	1	0
2	1	0	0	0
3	0	0	1	0

그래프 표현: 인접리스트(adjacency list)



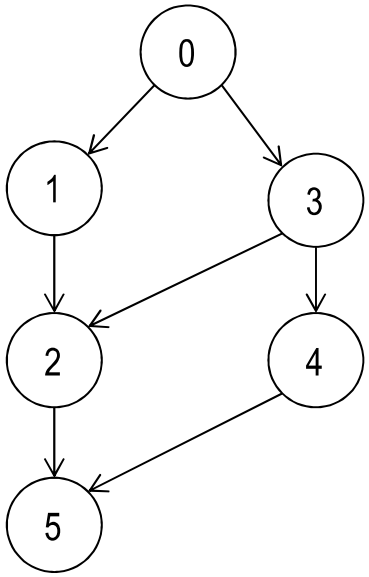


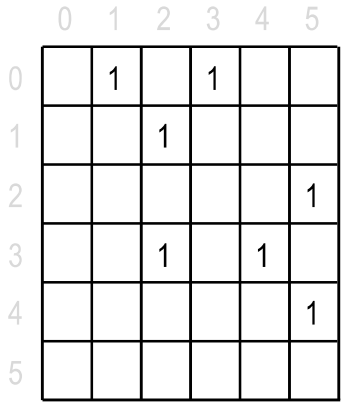


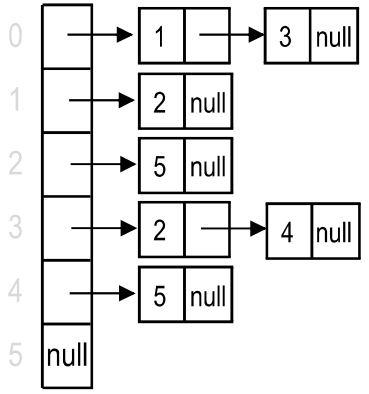


그래프 표현

V = { 0, 1, 2, 3, 4, 5 }
E = { <0,1>,<0,3>,<1,2>,<3,2>,<3,4>,<2,5>,<4,5> }





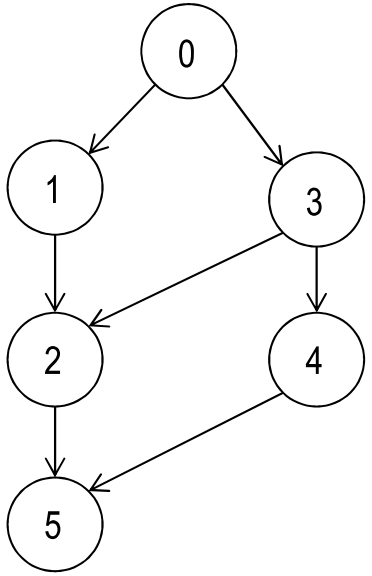


그래프 표현: 인접행렬

```
public class Test {
	public static void main(String[] args) {

		int	V=6;
		String	input="0 1 0 3 1 2 3 2 3 4 2 5 4 5";

		int	adjMat[][]=new int[V][V];

		String	s[]=input.split("\\s+");
		for (int i = 0; i < s.length; i+=2){
			int	v1=Integer.parseInt(s[i]);
			int	v2=Integer.parseInt(s[i+1]);
			adjMat[v1][v2]=1;
		}
		for (int i = 0; i < adjMat.length; i++) {
			for (int j = 0; j < adjMat[i].length; j++) {
				System.out.print(adjMat[i][j]);
			}
			System.out.println();
		}
	}
}
```

V = { 0, 1, 2, 3, 4, 5 }

E = { <0,1>,<0,3>,<1,2>,<3,2>,<3,4>,<2,5>,<4,5> }



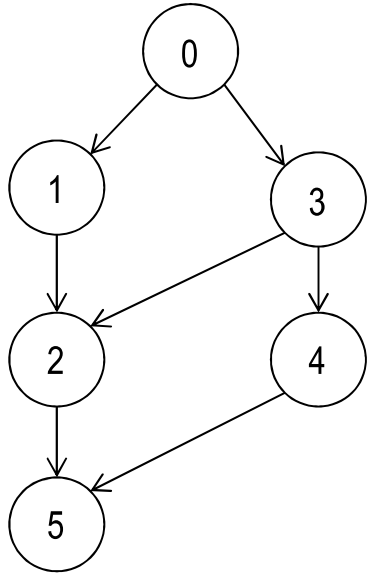

```
010100
001000
000001
001010
000001
000000
```

그래프 표현: 인접리스트

```
public class Test {
	public static void main(String[] args) {

		int	V=6;
		String	input="0 1 0 3 1 2 3 2 3 4 2 5 4 5";

		LinkedList<Integer> adjList[]=new LinkedList[V];
		for (int i = 0; i < adjList.length; i++){
			adjList[i]=new LinkedList<>();
		}
		String	s[]=input.split("\\s+");
		for (int i = 0; i < s.length; i+=2){
			int	v1=Integer.parseInt(s[i]);
			int	v2=Integer.parseInt(s[i+1]);
			adjList[v1].add(v2);
		}
		for (int i = 0; i < adjList.length; i++){
			System.out.println("node "+i+" => "+adjList[i]);
		}
	}
}
```

V = { 0, 1, 2, 3, 4, 5 }

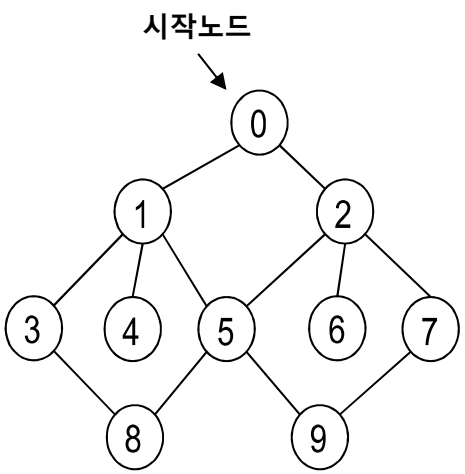
E = { <0,1>,<0,3>,<1,2>,<3,2>,<3,4>,<2,5>,<4,5> }




```
node 0 => [1, 3]
node 1 => [2]
node 2 => [5]
node 3 => [2, 4]
node 4 => [5]
node 5 => []
```

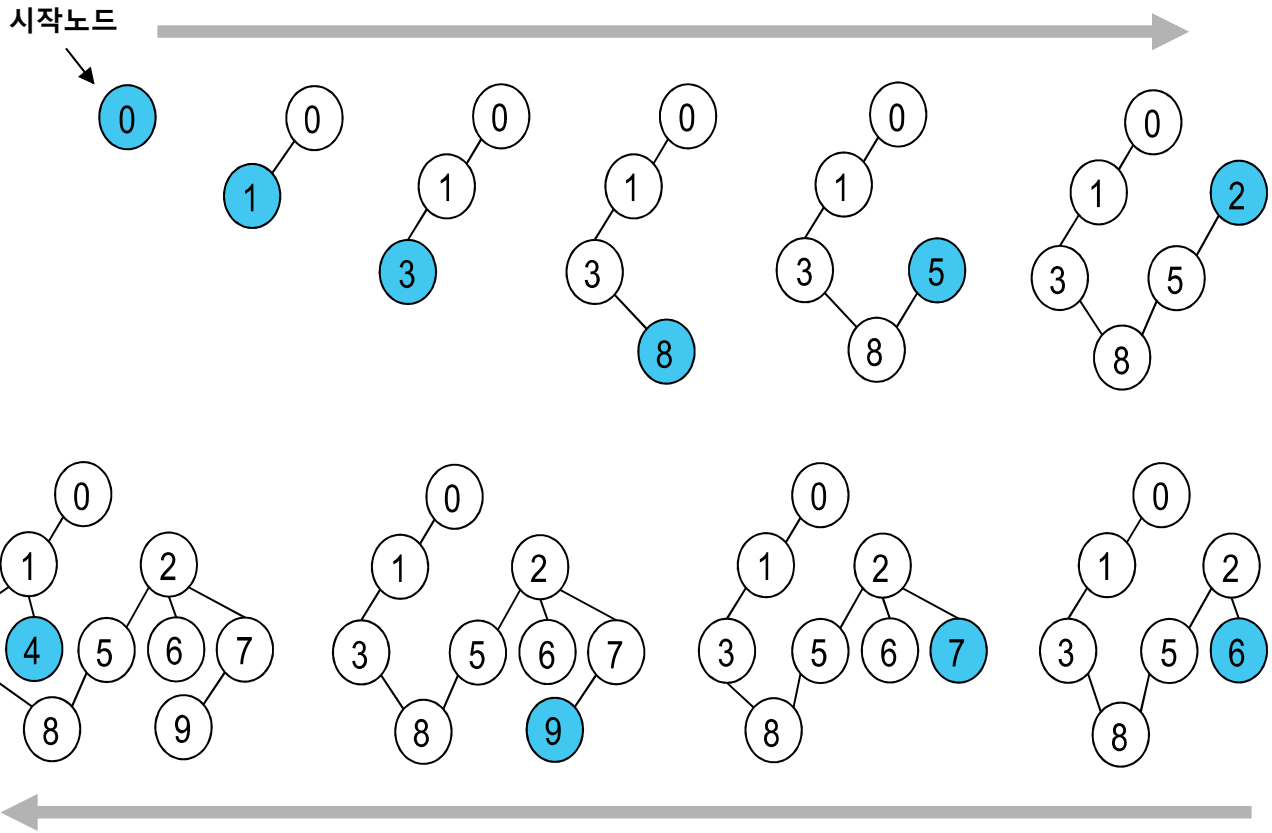
깊이우선탐색(depth first search, dfs)

- 깊이우선탐색 (dfs, depth first search)
  - 현재 노드를 방문(표시) 및 처리하고 현재 노드의 미방문 인접노드에 대해 깊이우선탐색(dfs)을 다시 적용한다.



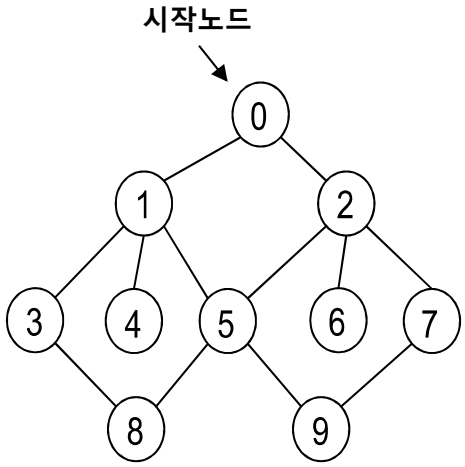
dfs 방문순서

0 1 3 8 5 2 6 7 9 4



깊이우선탐색(depth first search, dfs)

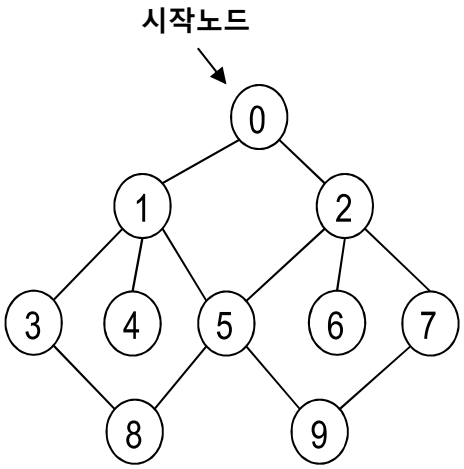
```
public class Test {
    public static void main(String[] args) {
        int     V=10; // 그래프 내 총 노드 개수 (노드 번호 0 ~ 9)
        String input="0 1 0 2 1 3 1 4 1 5 2 5 2 6 2 7 3 8 5 8 5 9 7 9";
        LinkedList<Integer> adjList[]=new LinkedList[V];
        for (int i = 0; i < adjList.length; i++) adjList[i]=new LinkedList<>();
        String s[]=input.split("\\s+");
        for (int i = 0; i < s.length; i+=2){
            int  v1=Integer.parseInt(s[i]), v2=Integer.parseInt(s[i+1]);
            adjList[v1].add(v2); // 무방향그래프로 처리
            adjList[v2].add(v1); // 무방향그래프로 처리
        }
        dfsVisit(adjList);
    }
    private static void dfsVisit(LinkedList<Integer>[] adjList) {
        boolean      visited[]=new boolean[adjList.length]; // adjList.length = 그래프 내 총 노드 개수
        dfs(adjList, visited, 0); // 시작노드부터 dfs 방문
    }
    private static void dfs(LinkedList<Integer>[] adjList, boolean[] visited, int v) {
        visited[v]=true;                    // 현재 노드 v 방문 표시
        System.out.print(v+" ");            // 현재 노드 v에 대한 처리
        for (Integer w : adjList[v]) {      // 현재 노드 v의 인접 노드 w에 대해
            if(visited[w]==false) {         // w가 미방문 인접 노드라면
                dfs(adjList, visited, w);   // 미방문 인접노드 w에 대해 dfs 재귀호출
            }
        }
    }
}
```



- **깊이우선탐색(dfs)**: 현재 노드를 방문(표시) 및 처리하고 현재 노드의 미방문 인접노드에 대해 깊이우선탐색을 다시 적용한다.

너비우선탐색(breadth first search, bfs)

- 너비우선탐색 (BFS, Breadth First Search)
  - 최초 시작 노드를 방문 표시 후 큐에 삽입
  - 공백 큐가 아닌 동안 아래 작업을 반복
    - 큐에서 노드 추출 후, 추출 노드의 인접 노드 중 미방문 노드를 방문 표시 후 큐에 삽입



큐

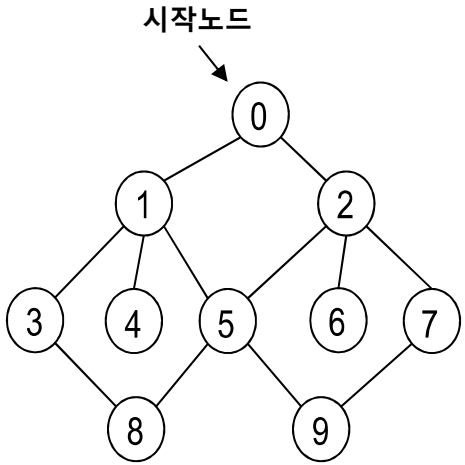
[]
[0]
[1, 2]
[2, 3, 4, 5]
[3, 4, 5, 6, 7]
[4, 5, 6, 7, 8]
[5, 6, 7, 8]
[6, 7, 8, 9]
[7, 8, 9]
[8, 9]
[9]
[]

방문순서

0 1 2 3 4 5 6 7 8 9

너비우선탐색(breadth first search, bfs)

```
public class Test {
    public static void main(String[] args) {
        int      V=10; // 그래프 내 총 노드 개수 (노드 번호 0 ~ 9)
        String  input="0 1 0 2 1 3 1 4 1 5 2 5 2 6 2 7 3 8 5 8 5 9 7 9";
        LinkedList<Integer> adjList[]=new LinkedList[V];
        for (int i = 0; i < adjList.length; i++) adjList[i]=new LinkedList<>();
        String  s[]=input.split("\\s+");
        for (int i = 0; i < s.length; i+=2){
            int  v1=Integer.parseInt(s[i]), v2=Integer.parseInt(s[i+1]);
            adjList[v1].add(v2); // 무방향그래프로 처리
            adjList[v2].add(v1); // 무방향그래프로 처리
        }
        bfsVisit(adjList);
    }
    private static void bfsVisit(LinkedList<Integer>[] adjList) {
        boolean        visited[]=new boolean[adjList.length]; // adjList.length = 그래프 내 총 노드 개수
        LinkedList<Integer>  queue=new LinkedList<>();
        visited[0]=true;        // 시작노드를 방문표시
        queue.addLast(0);    // 시작노드를 큐에 삽입
        while(queue.isEmpty()==false){            // 공백 큐가 아닌 동안
            int v=queue.removeFirst();            // 큐에서 노드 추출
            System.out.print(v+" ");              // 큐에서 추출된 노드 v에 대한 처리
            for (Integer w : adjList[v]) {        // 큐에서 추출된 노드 v의 인접노드 w에 대해
                if(visited[w]==false){            // w가 미방문 인접노드라면
                    visited[w]=true;              // 미방문 인접노드 w를 방문표시
                    queue.addLast(w);             // 미방문 인접노드 w를 큐에 삽입
                }
            }
        }
    }
}
```



- **너비우선탐색(bfs)**: 시작 노드를 방문(표시)하고 큐에 삽입한 다음, 공백 큐가 아닌 동안 *"큐에서 추출된 노드의 미방문 인접 노드들을 방문(표시)하고 큐에 삽입하는 작업"*을 반복한다.

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 프로그래밍 콘테스트 챌린징, Akiba 등 공저, 로드북, 2011.
- https://introcs.cs.princeton.edu/
- Introduction to Algorithms, Cormen et al., 3rd Edition (The MIT Press)