1. (**실습**: 그래프 표현, 인접행렬) 다음은 인접행렬에 기반한 방향그래프 표현을 구현한 예시 코드이다. 그래프의 총 노드 수는 변수 V에, 간선 집합은 문자열 변수 input에 저장되어 있다고 가정한다. input에 저장된 문자열 "0 1 0 3 1 2 ..."는 노드 0에서 노드 1로의 간선, 노드 0에서 노드 3으로의 간선, 노드 1에서 노드 2로의 간선 등이 존재함을 의미한다. 아래 코드를 입력하고 실행하면서 인접행렬 그래프 표현 구현법을 학습하시오.

   - **실습** #1: 이 코드는 방향그래프(directed graph)를 가정하고 input에 저장된 간선을 directed edge로 처리하였다. input에 저장된 간선 정보를 undirected edge로 처리하도록 아래 코드를 수정하시오.

   - 예제 그래프의 인접행렬

```
01010000
00100000
00000100
00101000
00000100
00000000
00000001
00000000
```

```
public class Test {
	public static void main(String[] args) {
		int		V=8;
		String	input="0 1 0 3 1 2 3 2 3 4 2 5 4 5 6 7";
		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();
		}
	}
}
```

2. (**실습**: 그래프 표현, 인접리스트) 다음은 인접리스트에 기반한 방향그래프 표현을 구현한 예시 코드이다. 그래프의 총 노드 수는 변수 V에, 간선 집합은 문자열 변수 input에 저장되어 있다고 가정한다. input에 저장된 문자열 "0 1 0 3 1 2 ..."는 노드 0에서 노드 1로의 간선, 노드 0에서 노드 3으로의 간선, 노드 1에서 노드 2로의 간선 등이 존재함을 의미한다. 아래 코드를 입력하고 실행하면서 인접리스트 그래프 표현 구현법을 학습하시오.

- **실습** #1: 이 코드는 방향그래프(directed graph)를 가정하고 input에 저장된 간선을 directed edge로 처리하였다. input에 저장된 간선 정보를 undirected edge로 처리하도록 아래 코드를 수정하시오.

- 예제 그래프의 인접리스트

```
node 0 => [1, 3]
node 1 => [2]
node 2 => [5]
node 3 => [2, 4]
node 4 => [5]
node 5 => []
node 6 => [7]
node 7 => []
```

```
public class Test {
	public static void main(String[] args) {
		int		V=8;
		String		input="0 1 0 3 1 2 3 2 3 4 2 5 4 5 6 7";
		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]);
		}
	}
}
```

3. (**실습**: 인접리스트 그래프 표현 및 DFS 방문) 다음은 인접리스트에 기반한 무방향그래프 표현을 사용하여 그래프 내 연결요소들을 dfs 방문하는 코드이다. 그래프의 총 노드 수는 변수 V에, 간선 집합은 문자열 변수 input에 저장되어 있다고 가정한다. input에 저장된 문자열 "0 1 0 3 1 2 ..."는 노드 0에서 노드 1로의 간선, 노드 0에서 노드 3으로의 간선, 노드 1에서 노드 2로의 간선 등이 존재함을 의미한다. 아래 코드를 입력하고 실행하면서 인접리스트 그래프 표현 및 dfs 탐색법을 학습하시오.

- **실습** #1: 아래 코드의 그래프는 연결요소가 2개이다. 그래프 내 연결요소의 총 개수를 출력하도록 아래 코드를 수정하시오.

```
public class Test {
	public static void main(String[] args) {
		String	input="0 1 0 3 1 2 3 2 3 4 2 5 4 5 6 7"; // 무방향 간선으로 해석
		int	V=8;

		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);
			adjList[v2].add(v1);
		}
		for (int i = 0; i < adjList.length; i++) System.out.println("node "+i+" => "+adjList[i]);
		connected(adjList, V);
	}
	private static void connected(LinkedList<Integer>[] adjList, int V) {
		boolean	visited[]=new boolean[V];
		for (int i = 0; i < visited.length; i++) {
			if(visited[i]==false){
				dfs(adjList, V, visited, i);
				System.out.println();
			}
		}
	}
	private static void dfs(LinkedList<Integer>[] adjList, int V, boolean[] visited, int v) {
		visited[v]=true;
		System.out.print(v+" ");
		for (Integer i : adjList[v]) {
			if(visited[i]==false) dfs(adjList, V, visited, i);
		}
	}
}
```

4. (**실습**: 인접행렬 그래프 표현 및 DFS 방문) 다음은 인접행렬에 기반한 무방향그래프 표현을 사용하여 그래프 내 연결요소들을 dfs 방문하는 코드이다. 그래프의 총 노드 수는 변수 V에, 간선 집합은 문자열 변수 input에 저장되어 있다고 가정한다. input에 저장된 문자열 "0 1 0 3 1 2 ..."는 노드 0에서 노드 1로의 간선, 노드 0에서 노드 3으로의 간선, 노드 1에서 노드 2로의 간선 등이 존재함을 의미한다.

- **실습** #1: 아래 dfs 함수는 인접행렬로 주어진 그래프에 대해 깊이우선탐색을 수행하여 방문 노드들을 출력한다. 이 함수를 구현하시오. 이전 문제에서처럼 dfs(int[][] adjMat, int V) 함수 내에서 dfs(int[][] adjMat, int V, boolean[] visited, int v) 함수를 호출하는 방식으로 구현하시오.

```
public class Test {
	public static void main(String[] args) {
		int		V=8;
		String	input="0 1 0 3 1 2 3 2 3 4 2 5 4 5 6 7"; // 무방향 간선으로 해석
		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;
			adjMat[v2][v1]=1;
		}
		dfs(adjMat, V);
	}
	private static void dfs(int[][] adjMat, int V) {
	}
	private static void dfs(int[][] adjMat, int V, boolean[] visited, int v) {
	}
}
```

5. (인접리스트 그래프 표현 및 BFS 방문) 다음은 인접리스트에 기반한 무방향그래프 표현을 사용하여 그래프 내 연결요소들을 bfs 방문하는 코드이다. 그래프의 총 노드 수는 변수 V에, 간선 집합은 문자열 변수 input에 저장되어 있다고 가정한다. input에 저장된 문자열 "0 1 0 3 1 2 ..."는 노드 0에서 노드 1로의 간선, 노드 0에서 노드 3으로의 간선, 노드 1에서 노드 2로의 간선 등이 존재함을 의미한다. 아래 코드를 입력하고 실행하면서 인접리스트 그래프 표현 및 bfs 탐색법을 학습하시오.

```
public class Test {
	public static void main(String[] args) {
		String	input="0 1 0 3 1 2 3 2 3 4 2 5 4 5 6 7"; // 무방향 간선으로 해석.
		int	V=8;

		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);
			adjList[v2].add(v1);
		}
		for (int i = 0; i < adjList.length; i++) System.out.println("node "+i+" => "+adjList[i]);
		connected(adjList, V);
	}

	private static void connected(LinkedList<Integer>[] adjList, int V) {
		boolean	visited[]=new boolean[V];
		for (int i = 0; i < visited.length; i++) {
			if(visited[i]==false){
				bfs(adjList, V, visited, i);
				System.out.println();
			}
		}
	}
	private static void bfs(LinkedList<Integer>[] adjList, int V, boolean[] visited, int v) {
		LinkedList<Integer>	queue=new LinkedList<>();
		visited[v]=true;
		queue.addLast(v);
		while(!queue.isEmpty()){
			v=queue.removeFirst();
			System.out.print(v+" ");
			for (Integer w : adjList[v]) {
				if(visited[w]==false){
					visited[w]=true;
					queue.addLast(w);
				}
			}
		}
	}
}
```

6. (**실습**: 그래프 응용) 새롭게 오픈된 SNS 서비스에 등록된 사용자는 총 10명(사용자 id는 0부터 9까지)이며, 아래 코드의 input은 현재까지 서로 친구 관계에 있는 사용자 id들을 문자열로 나열한 것이다. 예를 들어 input 문자열 첫 부분의 "0 1 2 3 ..."은 사용자 0과 사용자 1이 친구이며, 사용자 2과 사용자 3이 친구임을 각각 의미한다. 친구 관계로 연결된 사용자들은 같은 친구 그룹에 속한다고 가정한다. connected()는 각 사용자의 소속 그룹 id를 정수 배열로 반환하는 함수이다. 같은 그룹에 속한 사용자들은 그룹 id가 같다. isFriend(int[] groups, int i, int j)는 사용자 i와 사용자 j가 친구 관계인 경우 true를 그렇지 않은 경우 false를 반환하는 함수이다. 다음은 실행 결과이다.

- 실행결과:

```
[1, 1, 2, 2, 2, 3, 3, 3, 3, 3]
false
true
```

- **실습** #1: 아래 코드를 완성하시오.
- **실습** #2: 총 100명 사용자에 대한 다음 input 문자열 값에 대해 아래 코드의 실행 결과를 출력하시오.

```
int     V=100;
Stringinput="0 1 0 2 0 3 1 2 3 4 3 5 3 6 4 5 5 6 5 7 7 8 8 9 8 10 8 11 11 12 11 13 11 14 13 14 13 15 14 15 14 16 14 17
17 18 17 19 18 19 18 20 19 20 21 22 21 23 21 24 22 23 24 25 24 26 24 27 28 29 28 30 31 32 31 33 31 34 33 34 33 35
34 36 34 37 35 36 35 37 35 38 37 38 37 39 39 40 39 41 40 41 41 42 41 43 41 44 42 43 42 45 45 46 45 47 45 48 48 49
50 52 51 52 53 54 53 56 54 55 54 56 54 57 56 57 56 58 62 63 64 65 64 66 64 67 66 67 67 69 67 70 69 70 69 71 69 72
73 75 73 76 75 76 76 77 76 78 78 79 78 80 79 80 79 81 79 82 80 81 81 82 81 83 82 83 82 84 84 85 84 86 84 87 86 87
87 88 88 89 88 90 88 91 89 90 89 92 90 91 90 92 93 94 93 95 94 95 94 96 94 97 95 96 95 97 96 97 96 98 96 99";
```

- **실습** #3: 전체 사용자 수가 백만명이며, 친구관계 정보가 실시간으로 변하는 상황에서도 당신이 작성한 아래 코드는 효율적으로 동작할 수 있겠는가?

```
public class Test {
	public static void main(String[] args) {
		int		V=10;
		String	input="0 1 2 3 3 4 5 6 6 7 7 8 8 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]);
			int		v2=Integer.parseInt(s[i+1]);
			adjList[v1].add(v2);
			adjList[v2].add(v1);
		}
		int		groups[]=connected(adjList, V);
		System.out.println(Arrays.toString(groups));
		System.out.println(isFriend(groups,1,4));
		System.out.println(isFriend(groups,5,9));
		// input="0 1 2 3 3 4 5 6 6 7 7 8 8 9 4 5";
	}
}
```

7. (**실습**: BFS 경로 탐색) 다음은 bfs 기반 그래프 경로 탐색 코드과 그 실행 결과를 보인 것이다. (Reference: https://algs4.cs.princeton.edu/41graph/BreadthFirstPaths.java.html, GPLv3)

- 실행 결과

dist => [0.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 3.0, 3.0, 4.0]
prev => [-1, 0, 0, 0, 1, 1, 2, 3, 3, 4, 6, 9]

- **실습** #1: 시작노드 0으로부터의 나머지 각 노드로의 bfs 경로 정보(아래 예시 참조)가 출력되도록 아래 코드를 수정하시오.

0 ~ 0 => [0]
0 ~ 1 => [0, 1]
0 ~ 2 => [0, 2]
0 ~ 3 => [0, 3]
0 ~ 4 => [0, 1, 4]
0 ~ 5 => [0, 1, 5]
0 ~ 6 => [0, 2, 6]
0 ~ 7 => [0, 3, 7]
0 ~ 8 => [0, 3, 8]
0 ~ 9 => [0, 1, 4, 9]
0 ~ 10 => [0, 2, 6, 10]
0 ~ 11 => [0, 1, 4, 9, 11]

- **실습** #2: 아래 main 함수의 마지막 주석 처리된 두 문장은 정상 실행될 경우 다음과 같이 start 노드부터 end 노드까지의 bfs 경로 정보를 출력한다. 아래 주석 처리된 두 문장이 정상 실행되도록 하시오. bfs(adjList, V, start, end)는 start 노드부터 end 노드까지의 경로 정보만 출력하면 되는데 만약 start 노드부터 다른 모든 노드까지의 bfs 경로를 탐색한다면 그래프 내 노드 수가 많은 경우 비효율이 발생할 수 있다.

0 ~ 10 => [0, 2, 6, 10]

```
public class Test {
	public static void main(String[] args) {
		String		input="0 1 0 2 0 3 1 4 1 5 2 5 2 6 3 6 3 7 3 8 4 9 5 9 6 9 6 10 7 10 8 10 9 11 10 11";
		int		V=12;

		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);
			adjList[v2].add(v1);
		}
		bfs(adjList, V, 0);
		// int		start=0, end=10;
		// bfs(adjList, V, start, end);
	}
	private static void bfs(LinkedList<Integer>[] adjList, int V, int start){
		boolean		visited[]=new boolean[V];
		double		dist[]=new double[V];
		int		prev[]=new int[V];
		for (int i = 0; i < dist.length; i++) dist[i]=Double.MAX_VALUE;
		for (int i = 0; i < prev.length; i++) prev[i]=-1;
		LinkedList<Integer>		queue=new LinkedList<>();
		visited[start]=true;
		dist[start]=0;
		queue.addLast(start);
		while(!queue.isEmpty()){
			int		v=queue.removeFirst();
			for (Integer w : adjList[v]) {
				if(visited[w]==false){
					visited[w]=true;
					dist[w]=dist[v]+1;
					prev[w]=v;
					queue.addLast(w);
				}
			}
		}
		System.out.println("dist => "+Arrays.toString(dist));
		System.out.println("prev => "+Arrays.toString(prev));
```

```
    }
}
```

8. (**실습**: bfs 응용) 다음은 시작 노드로부터 특정 거리 이내 모든 노드들을 출력하는 코드이다. 아래 코드의 그래프의 경우, 노드 0으로부터 거리 0,1,2,3 이내 노드들은 각각 다음과 같다. 아래 코드의 bfs 함수를 완성하시오.

```
시작노드=0,  거리=0: 0
시작노드=0,  거리=1: 0 1 2 3
시작노드=0,  거리=2: 0 1 2 3 4 5 6 7 8
시작노드=0,  거리=3: 0 1 2 3 4 5 6 7 8 9 10
```

```
public class Test {
	public static void main(String[] args) {
		int		V=12;
		String	input="0 1 0 2 0 3 1 4 1 5 2 5 2 6 3 6 3 7 3 8 4 9 5 9 6 9 6 10 7 10 8 10 9 11 10 11";

		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);
			adjList[v2].add(v1);
		}
		boolean	visited[]=new boolean[V];
		int		start=0, distance=3;
		bfs(adjList, V, visited, start, distance);
	}
	private static void bfs(LinkedList<Integer>[] adjList, int V, boolean[] visited, int v, int distance) {
	}
}
```

9. (**실습**: 다익스트라 최단경로탐색 알고리즘) 다음은 다익스트라의 최단경로탐색 알고리즘을 구현한 코드이다. 그래프의 총 노드 수는 변수 V에, 가중치 부착 간선 집합은 문자열 변수 input에 저장되어 있다고 가정한다. input에 저장된 문자열 "0 1 11 0 3 22 1 2 99 ..."는 가중치 11을 갖는 노드 0에서 노드 1로의 간선, 가중치 22를 갖는 노드 0에서 노드 3으로의 간선, 가중치 99를 갖는 노드 1에서 노드 2로의 간선 등이 존재함을 의미한다. 아래 코드를 입력하고 실행하면서 다익스트라 최단경로탐색 알고리즘 구현법을 학습하시오. (Reference: https://en.wikipedia.org/wiki/Dijkstra's_algorithm, CC-BY-SA, Reference: p.299 in (Horowitz et al., 1993))

- **실습** #1: 아래 코드는 시작 노드로부터 그래프 내 각 노드까지의 최단경로 거리 값(dist)과 최단경로 상의 직전 노드 정보(prev)를 출력한다. 시작 노드로부터 그래프 각 노드까지의 최단경로를 출력(아래 실행 결과 예시 참조)하도록 아래 코드를 수정하시오. 아래 예시에서 [0,3,4,5]는 시작노드 0으로부터 노드 5까지의 최단경로를 의미하며, []는 경로가 존재하지 않음을 의미한다.

```
From 0 to 0 => [0]
From 0 to 1 => [0, 1]
From 0 to 2 => [0, 3, 2]
From 0 to 3 => [0, 3]
From 0 to 4 => [0, 3, 4]
From 0 to 5 => [0, 3, 4, 5]
From 0 to 6 => []
From 0 to 7 => []
```

- **실습** #2: 아래 코드의 인접행렬을 인접리스트로 대체하여 구현하시오.

```
public class Test {
	public static void main(String[] args) {
		String	input="0 1 11 0 3 22 1 2 99 3 2 11 3 4 33 2 5 88 4 5 44 6 7 55";
		int	V=8;
		double		adjMat[][]=new double[V][V];
		String	s[]=input.split("\\s+");
		for (int i = 0; i < s.length; i+=3){
			int	v1=Integer.parseInt(s[i]), v2=Integer.parseInt(s[i+1]);
			adjMat[v1][v2]=Double.parseDouble(s[i+2]);
		}
		DijkstraShortestPath(adjMat, V, 0);
	}
	private static void DijkstraShortestPath(double[][] adjMat, int V, int source) {
		boolean	found[]=new boolean[V];
		double	dist[]=new double[V];
		int	prev[]=new int[V];
		for (int i = 0; i < V; i++) {
			dist[i]=Double.MAX_VALUE;
			prev[i]=-1;
		}
		dist[source]=0;
		for (int i = 0; i < V; i++) {
			double	min=Double.MAX_VALUE;
			int	u=-1; // 최단거리 미결정 노드 중 최단거리 노드
			for (int v = 0; v < V; v++) if(!found[v] && dist[v]<min){ min=dist[v]; u=v; }
			if(u==-1) break;
			found[u]=true;
			for (int v = 0; v < V; v++) {
				if(found[v] || adjMat[u][v]<=0 || dist[v]<=dist[u]+adjMat[u][v]) continue;
				dist[v]=dist[u]+adjMat[u][v];
				prev[v]=u;
			}
		}
		System.out.println(Arrays.toString(dist));
		System.out.println(Arrays.toString(prev));
	}
}
```

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍 미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.
- 남궁성. 자바의 정석. 도우출판.
- 김윤명. (2010). 뇌를 자극하는 JSP & Servlet. 한빛미디어.