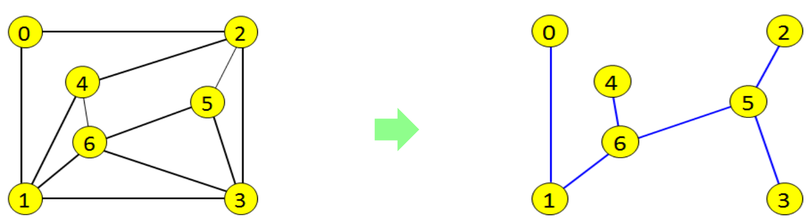
# Minimum Spanning Tree , MST

융합 IT 학과 최학준 이재근 지도교수 : 최현수 교수님

## I. 최소 신장 트리 ( Minimum Spanning Tree , MST )

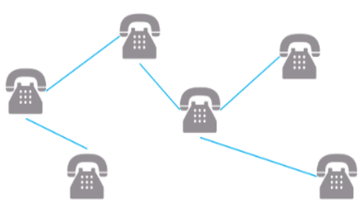
- MST : 하나의 연결성분으로 이루어진 無방향 가중치 그래프에서 간선 가중치의 합이 최소인 신장트리(Spanning Tree)이다. MST를 찾는 대표적인 알고리즘은 **Kruskal II , Prim Algorithm** 이 있다.



無방향 가중치 그래프 MST

## II.최소 신장 트리 ( MST ) 사용 사례

- 최소 신장 트리( MST )는 다양한 분야에서 응용되는 가장 기본적인 그래프 문제이다. 도로 건설, 전기 회로, 통신, 배관 설치 등에서 모든 것들이 연결되어 있으면서 연결 비용을 최소화 하려는 목적을 갖고 있다.



## III. Kruskal II Algorithm

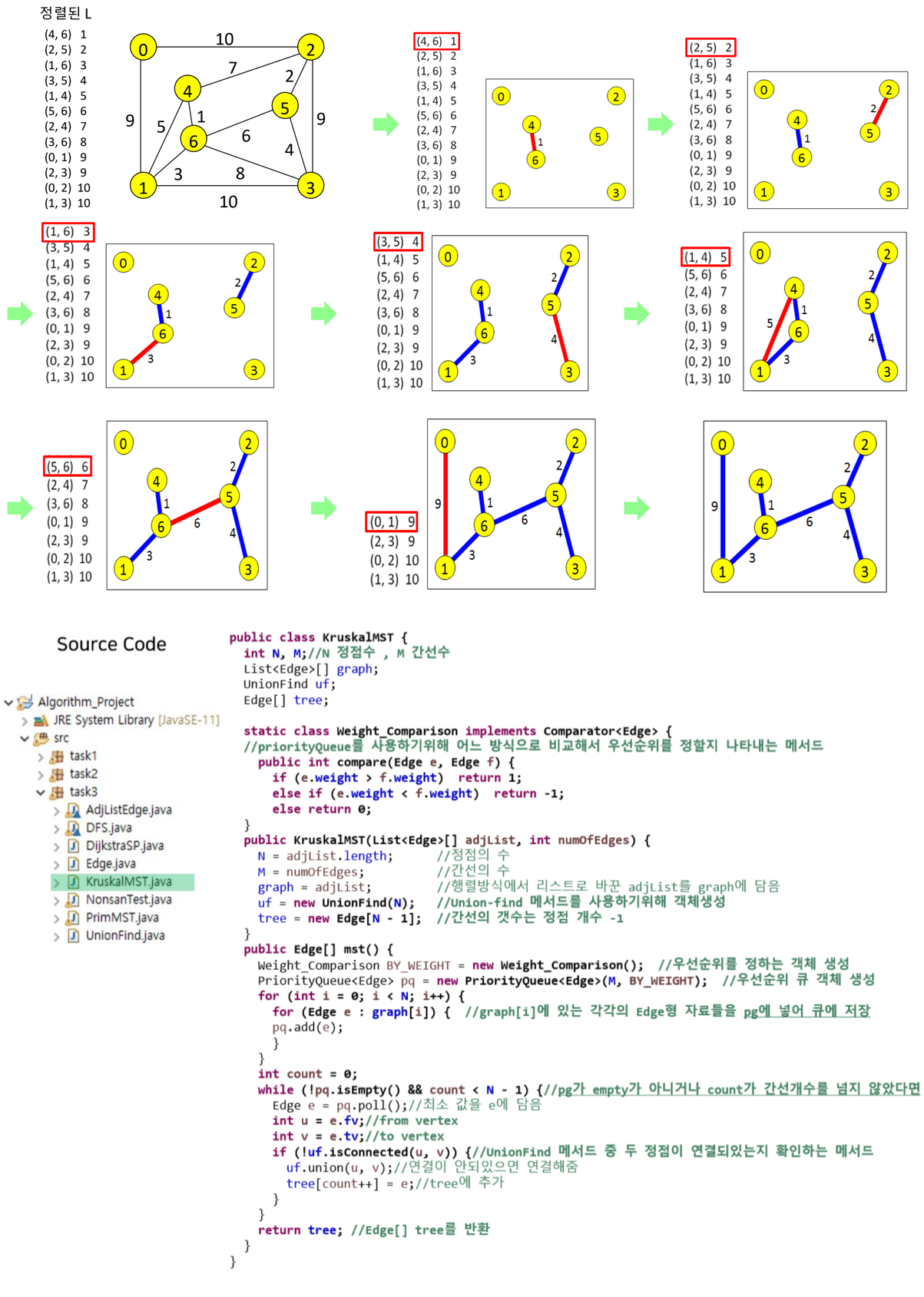
- **Pseudo Code**

```
Kruskal(N, E, cost) :
    sort edges in E by increasing cost
    while |T| < |N| − 1:
        let (u, v) be the next edge in E
        if u and v are on different components:
            join the components of u and v
            T = T ∪ {(u, v)}
    return T
```

1. 가중치가 감소하지 않는 순서로 간선리스트 L을 만든다.
2. While(트리의 간선 수 < N-1)
3. L에서 가장 작은 가중치를 가진 간선e를 가져오고, L에서 e를 제거.
4. If(간선e가 T에 추가하여 싸이클을 형성하지 않으면)
5. 간선 e를 T에 추가한다.

정렬된 L

(4, 6) 1
(2, 5) 2
(1, 6) 3
(3, 5) 4
(1, 4) 5
(5, 6) 6
(2, 4) 7
(3, 6) 8
(0, 1) 9
(2, 3) 9
(0, 2) 10
(1, 3) 10



Source Code



```
Algorithm_Project
  JRE System Library [JavaSE-11]
  src
    task1
    task2
    task3
      AdjListEdge.java
      DFS.java
      DijkstraSP.java
      Edge.java
      KruskalMST.java
      NonsanTest.java
      PrimMST.java
      UnionFind.java
```

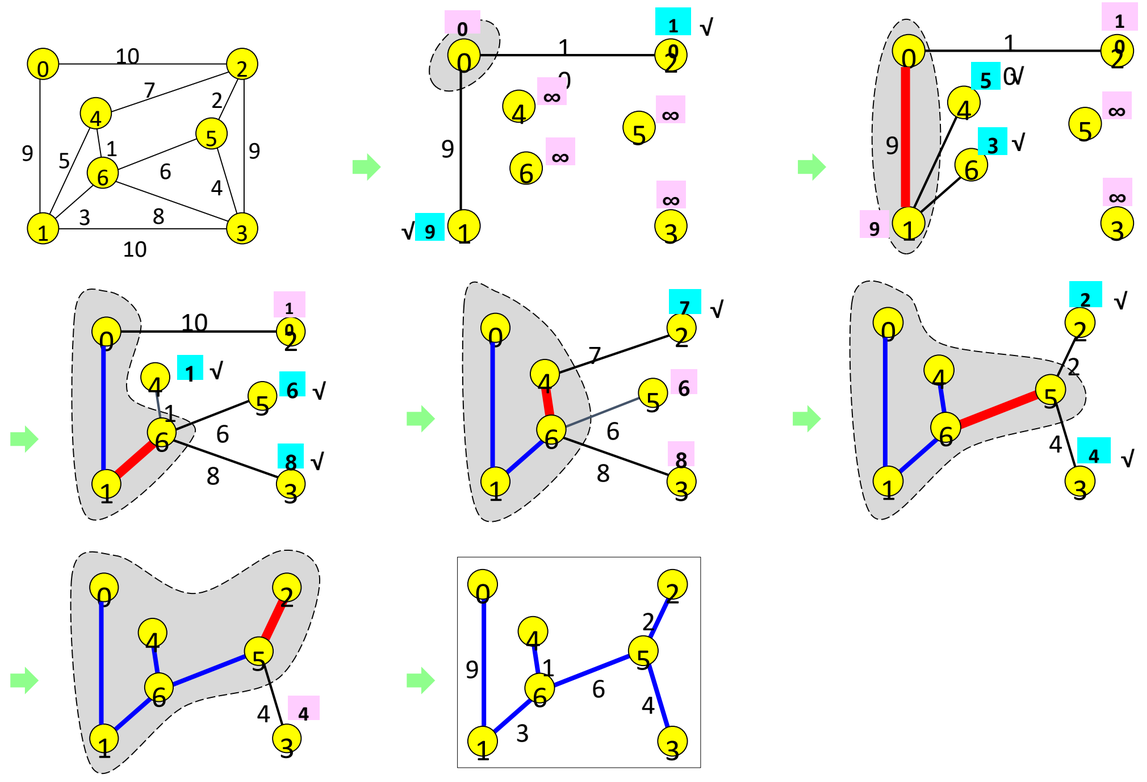
```
Edge[] tree;

static class Weight_Comparison implements Comparator<Edge> {
//priorityQueue를 사용하기위해 어느 방식으로 비교해서 우선순위를 정할지 나타내는 메서드
  public int compare(Edge e, Edge f) {
    if (e.weight > f.weight)  return 1;
    else if (e.weight < f.weight)  return -1;
    else return 0;
  }
public KruskalMST(List<Edge>[] adjList, int numOfEdges) {
  N = adjList.length;       //정점의 수
  M = numOfEdges;           //간선의 수
  graph = adjList;          //행렬방식에서 리스트로 바꾼 adjList를 graph에 담음
  uf = new UnionFind(N);    //Union-find 메서드를 사용하기위해 객체생성
  tree = new Edge[N - 1];  //간선의 갯수는 정점 개수 -1
}
public Edge[] mst() {
  Weight_Comparison BY_WEIGHT = new Weight_Comparison();  //우선순위를 정하는 객체 생성
  PriorityQueue<Edge> pq = new PriorityQueue<Edge>(M, BY_WEIGHT);  //우선순위 큐 객체 생성
  for (int i = 0; i < N; i++) {
    for (Edge e : graph[i]) {  //graph[i]에 있는 각각의 Edge형 자료들을 pg에 넣어 큐에 저장
    pq.add(e);
    }
  }
  int count = 0;
  while (!pq.isEmpty() && count < N - 1) {//pg가 empty가 아니거나 count가 간선개수를 넘지 않았다면
    Edge e = pq.poll();//최소 값을 e에 담음
    int u = e.fv;//from vertex
    int v = e.tv;//to vertex
    if (!uf.isConnected(u, v)) {//UnionFind 메서드 중 두 정점이 연결되있는지 확인하는 메서드
      uf.union(u, v);//연결이 안되있으면 연결해줌
      tree[count++] = e;//tree에 추가
    }
  }
  return tree; //Edge[] tree를 반환
  }
}
```

## IV. Prim Algorithm

- **Pseudo Code**

```
PRIM(G, w, r)
Q = ∅
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
    INSERT(Q, u)
DECREASE-KEY(Q, r, 0)        // r.key = 0
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            DECREASE-KEY(Q, v, w(u, v))
```

1. 배열Q를 ∞로 초기화한다. 시작정점 s의 D[s] = 0
2. while (T의 정점 수 < N)
3. T에 속하지 않은 각 정점 i에 대해 D[i]가 최소인 정점 minVertex를 찾아 T에 추가
4. for (T에 속하지 않은 각 정점 w에 대해서)
5. if (간선 (minVertex, w)의 가중치 < D[w])
6. D[w] = 간선 (minVertex, w)의 가중치



```
import java.util.List;
public class PrimMST {
    int N; // 그래프 정점의 수
    List<Edge>[] graph;

    public PrimMST(List<Edge>[] adjList) { // 생성자
        N = adjList.length;
        graph = adjList;
    }

    public int[] mst (int s) { // Prim 알고리즘,s는 시작정점
        boolean[] visited = new boolean[N]; // 방문된 정점은 true로
        int[] D = new int[N];
        int[] previous = new int[N]; // 최소신장트리의 간선으로 확정될 때 간선의 다른 쪽 (트리의)끝점
        for(int i = 0; i < N; i++){  // 초기화
            visited[i] = false;
            previous[i] = -1;
            D[i] = Integer.MAX_VALUE;  // D[i]를 최댓값으로 초기화
        }
        previous[s] = 0;  //시작정점 s의 관련 정보 초기화
        D[s] = 0;

        for(int k = 0; k < N; k++){ // 방문안된 정점들의 D 원소들중 에서 최솟값가진 정점 minVertex 찾기
            int minVertex = -1;
            int min = Integer.MAX_VALUE;
            for(int j=0;j<N;j++){
                if ((!visited[j])&&(D[j] < min)){
                    min = D[j];
                    minVertex = j;
                }
            }
            visited[minVertex] = true;
            for (Edge i : graph[minVertex]){ // minVertex에 인접한 각 정점의 D의 원소 갱신
                if (!visited[i.adjvertex]){  // 트리에 아직 포함 안된 정점이면
                    int currentDist = D[i.adjvertex];
                    int newDist = i.weight;
                    if (newDist < currentDist){
                        D[i.adjvertex] = newDist; // minVertex와 연결된 정점들의 D 원소 갱신
                        previous[i.adjvertex] = minVertex; // 트리 간선 추출을 위해
                    }
                }
            }
        }
        return previous; // 최소신장트리 간선 정보 리턴
    }
}
  public void printPrim(PrimMST a, int N) {
    int[] minTree = new int[graph.length - 1];
    int sum = 0;
    minTree = a.mst(0); // PrimMST형 변수에 출발지점을 넣고 그걸 minTree에 담음

    for (int i = 0; i < graph.length; i++) {
// previous에서 받은 tv를 adjList의 tv와 비교해 맞으면 fv값과 weight값을 가져옴.
      for (int j = 0; j < graph[i].size(); j++) {
        if (minTree[i] == graph[i].get(j).tv) {
          System.out.print("(" + graph[i].get(j).fv + "," + minTree[i] + "," + graph[i].get(j).weight + ") ");
          sum += graph[i].get(j).weight;
        }
      }
    }
    System.out.println("\n");
    System.out.println("최소신장트리의 간선 가중치 합은 " + sum + "입니다.");
  }
}
```

## V. 논산 주변의10개 도시의 연결도로망 그래프에 대한 MST