Data Structure

http://smartlead.hallym.ac.kr

Instructor: **Jin Kim**
010-6267-8189(033-248-2318)

jinkim@hallym.ac.kr
Office Hours:
Office : 자연대 7618

Lab(Graph)
최소비용신장트리

http://smart.hallym.ac.kr

Instructor: **Jin Kim**
010-6267-8189(033-248-2318)

jinkim@hallym.ac.kr
Office Hours:

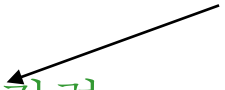
Minimum Spanning Tree

최소비용신장트리

늠름한 허스키

Kruskal's algorithm

Collections.sort()



1 (Sort the edges in an increasing order)//간선을 정렬
2 A:={}//간선들의집합
3 **While** E is not empty **do {//E는 정렬된간선집합**
3 take an edge (u, v) that is shortest in E E에서작은것부터꺼냄
and delete it from E
4 **if** u and v are in different components **then//사이클을 만들지 않으면**
add (u, v) to A // need cycle detection사이클탐지
}

Note: each time a shortest edge in E is considered.

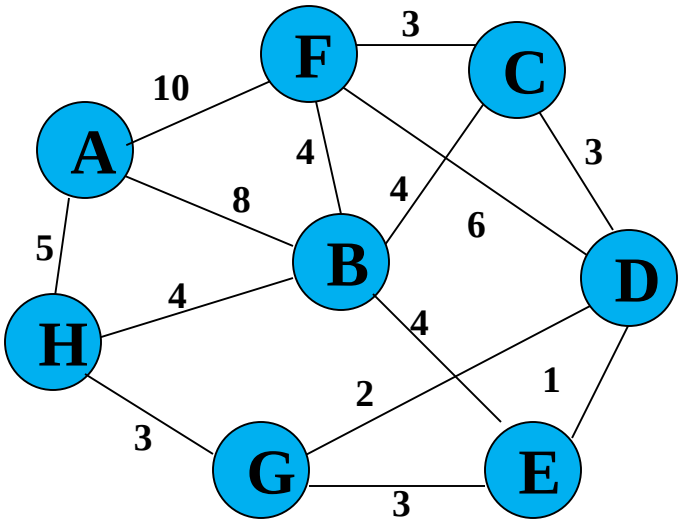
Disjoint Set(서로서 집합)

- 서로 중복되지 않는 부분 집합들.
  - 즉 공통 원소가 없는 부분집합들로 나누어진 원소들에 대한 자료구조
- Union-Find의 개념
  - Disjoint Set을 표현할 때 사용하는 알고리즘
- Union-Find의 연산
  - Make-set(x) : 초기화. X를 유일한 원소로 하는 새로운 집합을 만든다.
  - Union(x,y) : 합하기. X, y가 속한 집단을 합침.
  - Find(x) : 찾기. X가 속한 집합의 대표값(루트노드값)을 반환.

Union find

```
Initialize(int N)
  setsize = new int[N+1];
  parent = new int [N+1];
  for (int e=1; e <= N; e++)
    parent[e] = e;

int Find(int e)
  while (parent[e] != e)
    e = parent[e];
  return e;

Union(int i, int j)
    parent[i] = j;
```

```
int find(int x) { //find함수의 변종
            if (root[x] == x) {      return x;  } // 방번호와 같은 루트값을 가지면 x리턴
            else {      // "경로 압축(Path Compression)"
                        // find 하면서 만난 모든 값의 부모 노드를 root로 만든다.
                        return      root[x] = find(root[x]);
            }
}
```

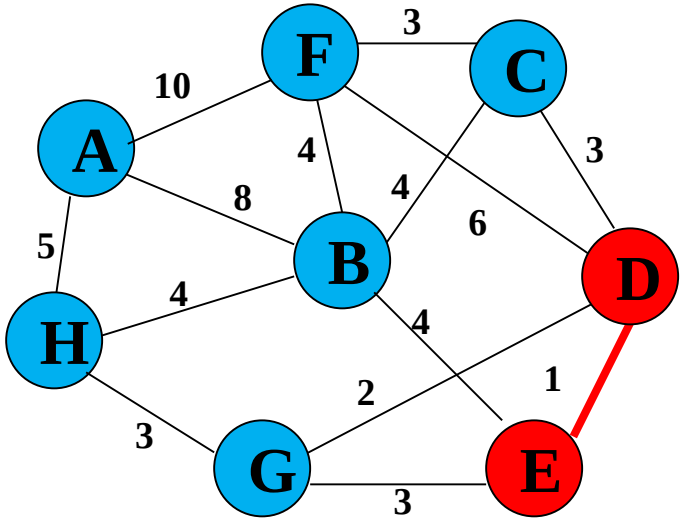


Initialize(초기화 일련번호부여)

vertex	*parent*
A(1)	1
B(2)	2
C(3)	3
D(4)	4
E(5)	5
F(6)	6
G(7)	7
H(8)	8

Parent(1)=1
Parent(2)=2
…

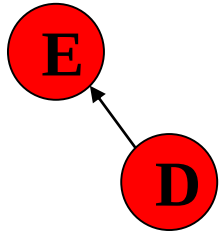
길이가 가장 작은 간선 (D, E)를 선택한다



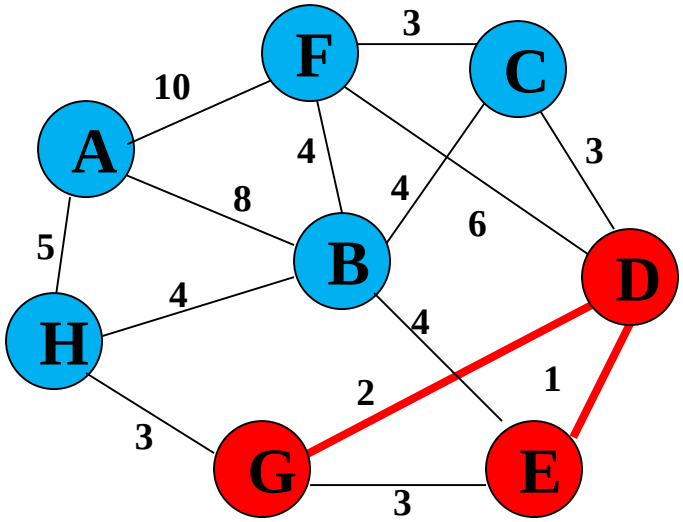
vertex	*parent*
A(1)	1
B(2)	2
C(3)	3
D(4)	4 → 5로변경
E(5)	5
F(6)	6
G(7)	7
H(8)	8

Find(D)=4 Find(E)=5

두 정점들의 루트가 다르다. 즉 두 정점은 서로서. 사이클안만든다
답의 일부로 저장하고 두 정점을 합침.
Union(D, E) 하자
Parent[4]=5 루트값이 큰 노드가 부모가 된다고 약속하자



길이가 가장 작은 간선 (D, G)를 선택한다

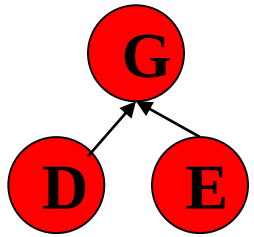


vertex	*parent*
A(1)	1
B(2)	2
C(3)	3
D(4)	5
E(5)	5
F(6)	6
G(7)	7
H(8)	8

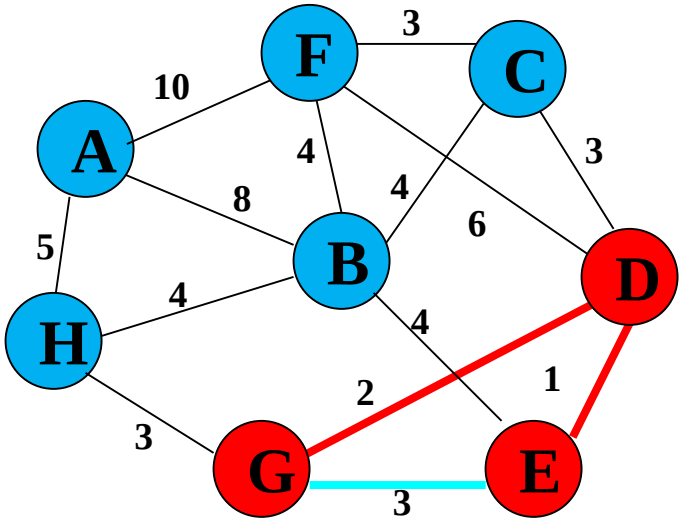
7로변경

Find(D)=5 Find(G)=7

두 정점들의 루트가 다르다. 즉 두 정점은 서로서. 사이클안만든다
답의 일부로 저장하고 두 정점을 합침.
Union(D, G)하면 D를 변경하도록 함. D의 현재부모가 E(5)이므로
Parent(E)를 7로변경, parent(D)=7로변경



길이가 가장 작은 간선 (G, E)를 선택한다



vertex	*parent*
A(1)	1
B(2)	2
C(3)	3
D(4)	7
E(5) ⟺	7
F(6)	6
G(7)	7
H(8)	8

Find(G)=7 Find(E)=7

서로소가 아니다. 두 정점사이의 간선은 사이클을 만들기 때문에 버린다.

Union find 알고리즘은 변종이 많다
우리는 다음 페이지처럼 간단한
버전을 사용한다.

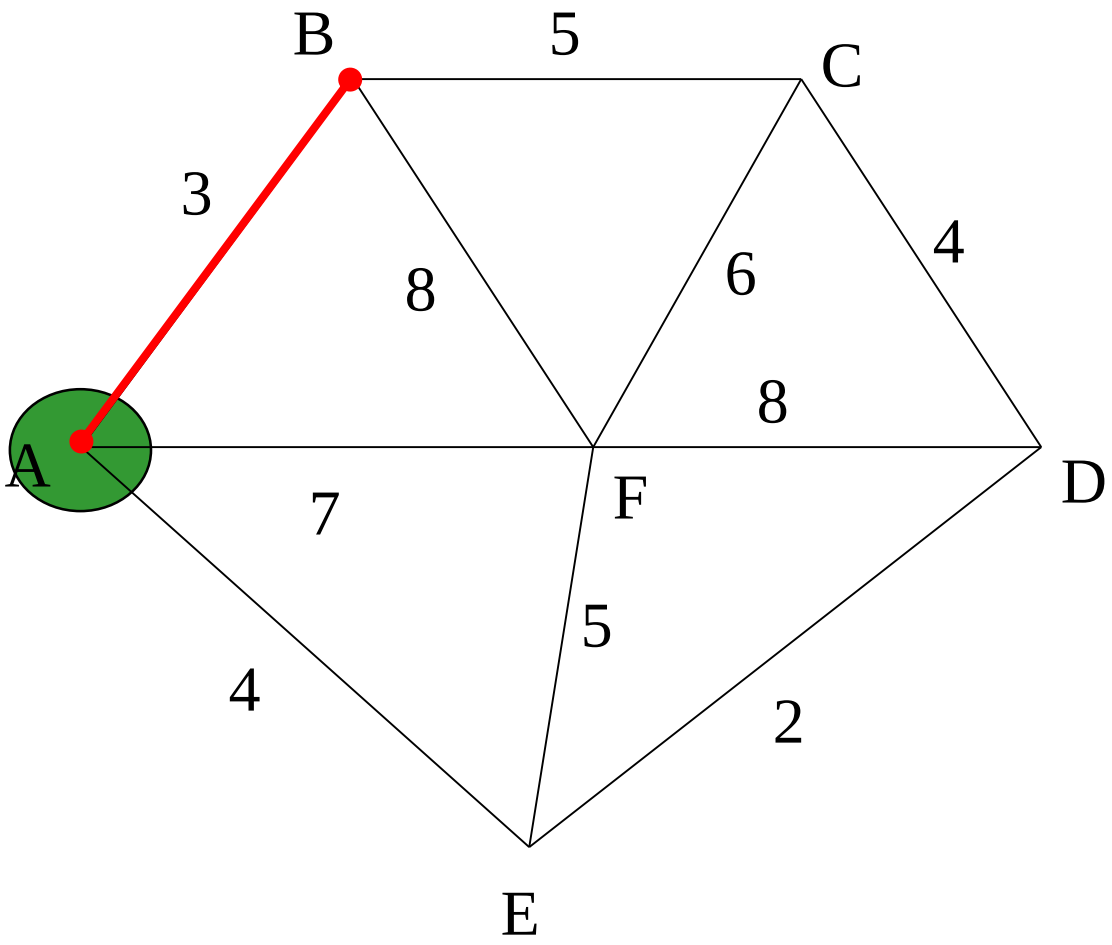
```java
public static void union(int x, int y) {
    x = find(x);
    y = find(y);

    if (x != y) {
        parent[y] = x;
    }
}

public static int find(int x) {  // 부모 노드 찾는 메소드
    if (parent[x] == x) {
        return x;
    }

    return parent[x] = find(parent[x]);
}

//same parent means cycle
public static boolean isSameParent(int x, int y) {
    x = find(x);  // find 메소드를 통해서 부모 노드 번호를 리턴 받음
    y = find(y);

    if (x == y) {
        return true;
    }
    else {
        return false;
    }
}
```

```java
//same parent means cycle
public static boolean isSameParent(int x, int y) {
    x = find(x);  // find 메소드를 통해서 부모 노드 번호를 리턴 받음
    y = find(y);

    if (x == y) {
        return true;
    }
    else {
        return false;
    }
}
```

```java
public static void main(String[] args) {
    edgeList = new ArrayList<Edge>();

    edgeList.add(new Edge(1, 7, 12));
    edgeList.add(new Edge(1, 4, 28));
    edgeList.add(new Edge(1, 2, 67));
    edgeList.add(new Edge(1, 5, 17));
    edgeList.add(new Edge(2, 4, 24));


    parent = new int[8];

    for (int i = 1; i <= 7; ++i) {
        parent[i] = i;
    }

    Collections.sort(edgeList);// 간선들을 정렬

    int sum = 0;
    for (int i = 0; i < edgeList.size(); ++i) {
        Edge edge = edgeList.get(i);
        if(!isSameParent(edge.v1, edge.v2)) {// no cycle
            System.out.println(" "+edge.v1+" "+edge.v2);
            sum += edge.cost;
            union(edge.v1, edge.v2);
        }
    }

    System.out.println(sum);
}
```

Prim's algorithm

Prim's Algorithm



Select any vertex

A

Select the shortest edge connected to that vertex

(3,4,7)

priorityQueue

Poll from the pq

AB 3

B not visited yet

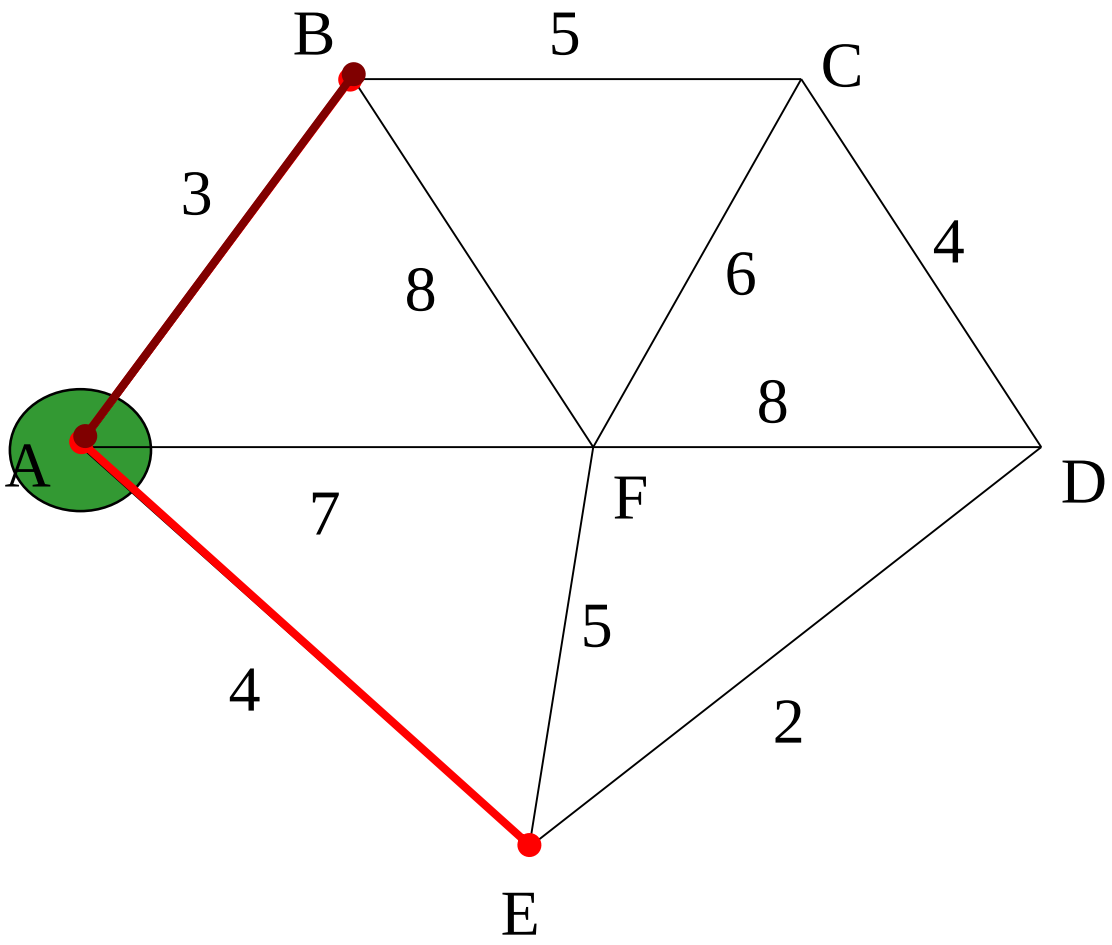
Visited[B]=true

Queue<-B

Mst=mst+(AB)

Pq<-5,8

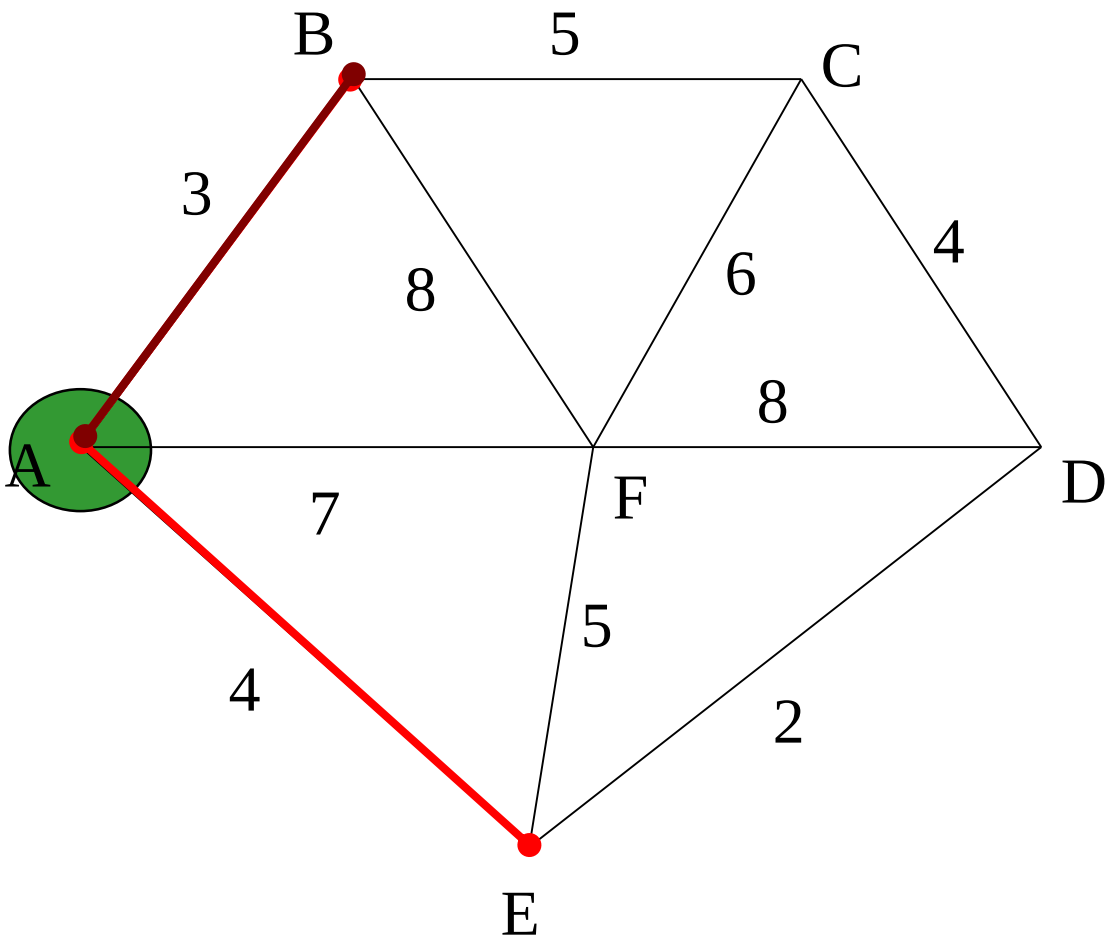
Prim's Algorithm



Prioirty queue
(4,5,7,8)

Poll from the pq
AE 4
E not visited yet
Visited[E]=true
Queue<-E
Mst=mst+(AE)
Pq<-5,2추가

Prim's Algorithm



Prioirty queue
(2, 5, 5,7,8)

Poll from the pq
ED 2
D not visited yet
Visited[D]=true
Queue<-D
Mst=mst+(ED)

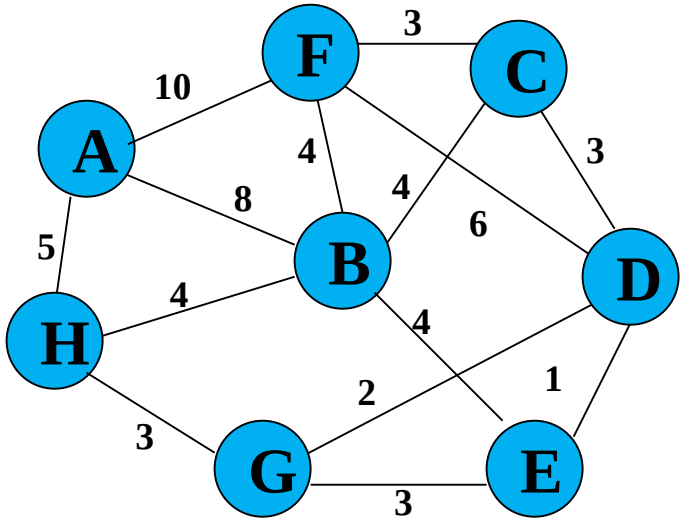
```java
public static void main(String[] args) {
    V = 9;
    E = 14;
    visited = new boolean[V + 1];
    mst = new ArrayList<>();
    graph = new Graph(V);

    graph.AddEdge(1, 2, 4);
    graph.AddEdge(2, 3, 8);
    graph.AddEdge(3, 4, 7);

    Prim();

    for (Edge edge : mst)
        System.out.println(edge.from + " - " + edge.to + " cost : " + edge.cost);

    System.out.println(min);
}
```

```java
public static void Prim() {
    PriorityQueue<Edge> pq = new PriorityQueue<>(); // 가중치가 낮은 순대로 간선을 정렬할 우선순위
    Queue<Integer> queue = new LinkedList<>();      // 정점방문 스케줄 처리를 위한 큐

    queue.add(1);                                   // 정점 1을 시작정점으로 선택

    while (!queue.isEmpty()) {
        int from = queue.poll();
        visited[from] = true;

        for (Edge edge : graph.edge[from]) {        // 현재 정점 from과 연결된 간선 중
            if (!visited[edge.to]) {                // 아직 정점 to를 방문하지 않았다면
                pq.add(edge);                       // 우선순위 큐에 간선을 추가한다.
            }
        }

        while (!pq.isEmpty()) {
            Edge edge = pq.poll();                  // 가중치가 가장 적은 간선이 나올 것이며,
            if (!visited[edge.to]) {                // 간선이 연결된 정점 to를 방문한 적이 없다면,
                queue.add(edge.to);                 // 큐에 삽입하여 다음에 방문한다.
                visited[edge.to] = true;            // 방문했던 정점을 다시 방문하지 않도록 방문표시.
                mst.add(edge);                      // 최소 스패닝 트리를 구성하는 간선 추가.
                min += edge.cost;                   // 총 최소 가중치 합을 구하기 위해 덧셈.
                break;
            }
        }
    }
}
```

```java
class Graph {
    List<Edge>[] edge;

    public Graph(int V) {
        edge = new LinkedList[V + 1];

        for (int i = 1; i <= V; i++)
            edge[i] = new LinkedList<>();
    }

    // 양방향 그래프
    public void AddEdge(int from, int to, int cost) {
        edge[from].add(new Edge(from, to, cost));
        edge[to].add(new Edge(to, from, cost));
    }
}
```

```java
class Edge implements Comparable<Edge> {
    int from, to, cost;

    public Edge(int from, int to, int cost) {
        this.from = from;
        this.to = to;
        this.cost = cost;
    }

    @Override // Prioirty Queue 우선순위 큐를 사용하기 위한 함수 오버라이딩
    public int compareTo(Edge e) {
        return this.cost - e.cost;
    }
}
```

프림의 알고리즘과 크루스칼의 알고리즘을 구현하라. 다음 그래프에 적용하라.



Difficult?어렵다?

- 자바에 대한 내공이 약하다.

2. Kruskal.java

위의 프로그램을 제출하라.