그리디 알고리즘

1. 최소 신장 트리 - 크루스칼 알고리즘
2. 최소 신장 트리 - 프림 알고리즘
3. 다익스트라 알고리즘

1. 최소 신장 트리 - 크루스칼 알고리즘

> **1)** 크기가 가장 작은 간선부터 모든 간선을 살핀다.
>
> 이때 간선은 가중치에 대해 오름차순으로 정렬되어 있다.
>
> **2)** 간선을 그래프에 포함 했을 때, MST에 사이클이 생기지 않으면 추가한다. 이 과정은 유니온 - 파인드 알고리즘을 이용한다.
>
> 정점 v와 정점 w를 잇는 간선e 가 있을 때, 정점 v와 정점 w가 같은 부모 노드를 가진다면 간선 e는 MST에 추가하지 않는다.

```
public class Main {
  // 유니온
  public static void union(int[] parent, int x, int y) {
    x = find(parent, x);
    y = find(parent, y);

    if(x < y) parent[y] = x;
    else parent[x] = y;
  }
    // 파인드
  public static int find(int[] parent, int x) {
    if(parent[x] == x) return x;
    else return find(parent, parent[x]);
  }

    // 크루스칼
  public static void kruskal(int[][] graph, int[] parent) {
    int cost = 0;
    for(int i = 0; i < graph.length;i++) {
      if (find(parent, graph[i][0]) != find(parent, graph[i][1])) {
        cost += graph[i][2];
        union(parent, graph[i][0], graph[i][1]);
      }
    }

        // 최소 신장 트리의 총 가중치 출력
    System.out.println(cost);
```

```
  }

  public static void main(String[] args) throws IOException {
      // 간선 입력 받기, 그래프에 저장
    BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
    int n = Integer.parseInt(bf.readLine());
    int m = Integer.parseInt(bf.readLine());
    int[][] graph = new int[m][3];

    StringTokenizer st;
    for (int i = 0; i < m; i++) {
      st = new StringTokenizer(bf.readLine());
      graph[i][0] = Integer.parseInt(st.nextToken()); // 간선 나가는 정점
      graph[i][1] = Integer.parseInt(st.nextToken()); // 간선 들어오는 정점
      graph[i][2] = Integer.parseInt(st.nextToken()); // 가중치
    }

        // 간선 정렬
    Arrays.sort(graph, (o1, o2) -> o1[2] - o2[2]);

        // 부모노드 초기화
    int[] parent = new int[n + 1];
    for (int i = 0; i < parent.length; i++) {
      parent[i] = i;
    }

    // 크루스칼 알고리즘 수행
    kruskal(graph, parent)
  }
}
```

```
입력
5
6
1 3 3
1 4 8
4 5 9
1 2 10
2 3 13
2 5 14

출력 결과
30
```

---

2. 최소 신장 트리 - 프림 알고리즘

> 프림 알고리즘은 시작 정점을 정해 우선 순위 큐에 넣는다.

> 우선 순위 큐에는 (`정점` , `가중치`) 형식으로 저장되며, 첫 시작은 (`시작 정점` , `0`)으로 넣는다.

> 우선 순위 큐가 빌 때까지 아래를 반복한다.
>
> **1)** 우선 순위 큐에서 하나를 꺼낸다. 꺼낸 정점을 `v` 라고 한.
>
> **2)** `v` 가 이미 MST에 포함됐다면 **1)**로 돌아간다. 그렇지 않다면 아래를 진행한다.
>
> **3)** `v` 와 연결된 간선을 모두 살핀다. 간선 (`w` , `cost`)는 `v` 와 `정점 w` 사이 연결된 간선이며 `cost` 가중치를 가진다. 만약 `w` 를 방문하지 않았다면 우선순위 큐에 추가한다.

```
// 간선 저장 위한 클래스
class Edge implements Comparable<Edge>{
  int w; // 간선 들어오는 정점
  int cost; // 간선 가중치

  Edge(int  w, int cost){
    this.w = w;
    this.cost = cost;
  }

  // 간선 오름차순으로 정렬
  @Override
  public int compareTo(Edge o) {
    return this.cost - o.cost;
  }
}

public class prim_main {
  static List<Edge>[] graph;

  public static void prim(int start, int n) {
    boolean[] visit = new boolean[n + 1];

    PriorityQueue<Edge> pq = new PriorityQueue<>();
    pq.offer(new Edge(start, 0));

    int total = 0;
    while(!pq.isEmpty()) {
      Edge edge = pq.poll();
      int v = edge.w;
      int cost = edge.cost;

      //방문 했으면 건너뜀
      if(visit[v]) continue;
```

```
      visit[v] = true;
      total += cost;

      for(Edge e : graph[v]) {
        if(!visit[e.w]) {
          pq.add(e);
        }
      }
    }

    // 완성된 최소 신장 트리의 총 가중치 합 출력
    System.out.println(total);
  }


  public static void main(String[] args) throws IOException {
   // 그래프 입력, 저장
    BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
    int n = Integer.parseInt(bf.readLine());
    int m = Integer.parseInt(bf.readLine());

    // 그래프 선언, 간선 리스트로 표현
    graph = new ArrayList[n + 1];
    for (int i = 0; i < graph.length; i++) graph[i] = new ArrayList<>();

    StringTokenizer st;
    for (int i = 0; i < m; i++) {
      st = new StringTokenizer(bf.readLine());
      int v = Integer.parseInt(st.nextToken());
      int w = Integer.parseInt(st.nextToken());
      int cost = Integer.parseInt(st.nextToken());

      graph[v].add(new Edge(w, cost));
      graph[v].add(new Edge(v, cost));
    }

    // 프림 알고리즘 수행
    prim(1, n);
  }
}
```

```
입력
5
6
1 3 3
1 4 8
4 5 9
1 2 10
2 3 13
2 5 14

출력 결과
30
```

3. 다익스트라 알고리즘

1753번: 최단경로

BAEKJOON ONLINE JUDGE

https://www.acmicpc.net/problem/1753

```
import java.util.*;
import java.io.*;

public class Main {

    static class Node{
        int v; //간선
        int cost; //가중치

        public Node(int v, int cost) {
            this.v = v;
            this.cost = cost;
        }
    }

    //각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트
    static ArrayList<Node>[] graph;
    //방문한 적이 있는지 체크하는 목적의 리스트
    static boolean[] visit;
    //최단 거리 테이블
    static int[] dist;

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer(br.readLine());

        int v = Integer.parseInt(st.nextToken());
        int e = Integer.parseInt(st.nextToken());
        int k = Integer.parseInt(br.readLine());

        graph = new ArrayList[v + 1];
        dist = new int[v + 1];
        visit = new boolean[v + 1];

        for (int i = 1; i <= v; i++) {
            graph[i] = new ArrayList<>();
            dist[i] = Integer.MAX_VALUE; //최대값으로 초기화, 최단거리를 찾기 위함.
        }

        for (int i = 0; i < e; i++) {
            // u -> v 로 가는 가중치 w가 주어진다.
            st = new StringTokenizer(br.readLine());
            int inputU = Integer.parseInt(st.nextToken());
            int inputV = Integer.parseInt(st.nextToken());
```

```
            int inputW = Integer.parseInt(st.nextToken());

            graph[inputU].add(new Node(inputV, inputW));
        }

        //다익스트라 알고리즘 수행
        dijkstra(k);

        for (int i = 1; i <= v; i++) {
            System.out.println(dist[i] == Integer.MAX_VALUE ? "INF" : dist[i]);
        }
    }

    static void dijkstra(int start) {
        //우선 순위 큐 사용, 가중치를 기준으로 오름차순한다.
        PriorityQueue<Node> q = new PriorityQueue<>((o1, o2) -> o1.cost - o2.cost);
        //시작 노드에 대해서 초기화
        q.add(new Node(start, 0));
        dist[start] = 0;

        while (!q.isEmpty()) {
            //현재 최단 거리가 가장 짧은 노드를 꺼내서 방문 처리 한다.
            Node now = q.poll();

            if (!visit[now.v]) {
                visit[now.v] = true;
            }

            for (Node next : graph[now.v]) {

                //방문하지 않았고, 현재 노드를 거쳐서 다른 노드로 이동하는 거리가 더 짧을 경우
                if (!visit[next.v] && dist[next.v] > now.cost + next.cost) {
                    dist[next.v] = now.cost + next.cost;
                    q.add(new Node(next.v, dist[next.v]));
                }
            }
        }
    }
}
```

```
입력
5 6
1
5 1 1
1 2 2
1 3 3
2 3 4
2 4 5
3 4 6

출력
0
2
3
```

```
7
INF
```