< 그래프 알고리즘 >

[A] 크루스칼 (유니언-파인드)

< 입력 예 >

```
8 16
4 5 35
4 7 37
5 7 28
0 7 16
1 5 32
0 4 38
2 3 17
1 7 19
0 2 26
1 2 36
1 3 29
2 7 34
6 2 40
3 6 52
6 0 58
6 4 93
```

< 출력 예 >

```
181
```

☞ **간선을 가중치로 정렬하는 대신 우선순위큐(최소힙) 사용하는 코드**

1) Python

```python
import heapq

class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])  # 경로 압축
        return self.parent[x]

    def union(self, a, b):
        p = self.find(a)
        q = self.find(b)
        if p != q:
            if self.rank[p] < self.rank[q]:
                self.parent[p] = q
            elif self.rank[p] > self.rank[q]:
                self.parent[q] = p
            else:
                self.parent[q] = p
                self.rank[p] += 1

def kruskal(n, edges):
    # 간선들을 우선순위 큐로 변환
    heapq.heapify(edges)
    uf = UnionFind(n)
```

```
    min_cost = 0
    edges_used = 0

    while edges and edges_used < n - 1:
        w, u, v = heapq.heappop(edges)
        if uf.find(u) != uf.find(v):
            uf.union(u, v)
            min_cost += w
            edges_used += 1

    return min_cost if edges_used == n - 1 else -1  # 신장 트리가 존재하지 않으면 -1 반환

# 입력 처리
n, m = map(int, input().split())
edges = []
for _ in range(m):
    u, v, w = map(int, input().split())
    edges.append((w, u, v))  # 가중치를 맨 앞에

print(kruskal(n, edges))
```

2) C++

```
#include <iostream>
#include <vector>
#include <queue>
#include <tuple>

using namespace std;

// Union-Find 자료 구조
class UnionFind {
public:
    UnionFind(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);  // 경로 압축
        }
        return parent[x];
    }

    void unionSets(int a, int b) {
        int p = find(a);
        int q = find(b);
        if (p != q) {
            if (rank[p] < rank[q]) {
                parent[p] = q;
            }
```

```
            else if (rank[p] > rank[q]) {
                parent[q] = p;
            }
            else {
                parent[q] = p;
                rank[p]++;
            }
        }
    }

private:
    vector<int> parent;
    vector<int> rank;
};

int kruskal(int n, vector<tuple<int, int, int>>& edges) {
    // 가중치를 기준으로 간선을 정렬하는 대신 우선순위 큐를 사용
    priority_queue<tuple<int, int, int>, vector<tuple<int, int, int>>,
        greater<tuple<int, int, int>>> pq(edges.begin(), edges.end());

    UnionFind uf(n);
    int min_cost = 0;
    int edges_used = 0;

    while (!pq.empty() && edges_used < n - 1) {
        tuple<int, int, int> e = pq.top();
        int w = get<0>(e);
        int u = get<1>(e);
        int v = get<2>(e);
        pq.pop();

        if (uf.find(u) != uf.find(v)) {
            uf.unionSets(u, v);
            min_cost += w;
            edges_used++;
        }
    }

    return (edges_used == n - 1) ? min_cost : -1;    //신장트리가 존재하지 않으면 -1 반환
}

int main() {
    int n, m;
    cin >> n >> m;

    vector<tuple<int, int, int>> edges;
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        edges.emplace_back(w, u, v);    //가중치를 맨 앞에!!! (우선순위큐)
    }

    cout << kruskal(n, edges) << endl;

    return 0;
}
```

3) Java

```
import java.util.*;

class UnionFind {
    private int[] parent;
    private int[] rank;

    public UnionFind(int n) {
        parent = new int[n];
        rank = new int[n];
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
            rank[i] = 0;
        }
    }

    public int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);  // 경로 압축
        }
        return parent[x];
    }

    public void unionSets(int a, int b) {
        int p = find(a);
        int q = find(b);
        if (p != q) {
            if (rank[p] < rank[q]) {
                parent[p] = q;
            } else if (rank[p] > rank[q]) {
                parent[q] = p;
            } else {
                parent[q] = p;
                rank[p]++;
            }
        }
    }
}

public class Main {

    public static int kruskal(int n, List<int[]> edges) {
        // 가중치를 기준으로 간선을 정렬하는 대신 우선순위 큐를 사용
        PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(e -> e[0]));

        pq.addAll(edges);

        UnionFind uf = new UnionFind(n);
        int min_cost = 0;
        int edges_used = 0;

        while (!pq.isEmpty() && edges_used < n - 1) {
            int[] edge = pq.poll();
```

```
            int w = edge[0];
            int u = edge[1];
            int v = edge[2];

            if (uf.find(u) != uf.find(v)) {
                uf.unionSets(u, v);
                min_cost += w;
                edges_used++;
            }
        }

        return (edges_used == n - 1) ? min_cost : -1;  // 신장트리가 존재하지 않으면 -1 반환
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();

        List<int[]> edges = new ArrayList<>();
        for (int i = 0; i < m; ++i) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            int w = sc.nextInt();
            edges.add(new int[]{w, u, v});  // 가중치를 맨 앞에!!! (우선순위큐)
        }

        System.out.println(kruskal(n, edges));
    }
}
```

[B] 다익스트라

< 입력 예 >

```
8 16 0
0 1 5
0 4 9
0 7 8
1 2 12
1 3 15
1 7 4
2 3 3
2 6 11
3 6 9
4 5 4
4 6 20
4 7 5
5 2 1
5 6 13
7 5 6
7 2 7
```

< 출력 예 >

```
0 : 0
1 : 5
2 : 14
3 : 17
4 : 9
5 : 13
6 : 25
7 : 8
```

☞ 입력 첫 줄은 정점의 수, 간선의 수, 출발정점이고, 다음 줄부터는 방향 그래프 간선 정보

☞ 출력은 각 정점까지의 최단 거리

1) Python

```python
import heapq

INF = int(1e9)

def dijkstra(graph, start, n):
    pq = [(0, start)]
    dist = [INF] * n
    dist[start] = 0

    while pq:
        distance, i = heapq.heappop(pq)

        if distance > dist[i]:  continue  #이 코드를 추가하면 더 효율적

        for j, w in graph[i]:
            if dist[i] + w < dist[j]:
                dist[j] = dist[i] + w
                heapq.heappush(pq, (dist[j], j))

    return dist

if __name__ == "__main__":
    n, m, st = map(int, input().split())

    graph = [[] for _ in range(n)]
    for _ in range(m):
        u, v, w = map(int, input().split())
        graph[u].append((v, w))

    dist = dijkstra(graph, st, n)

    for i in range(n):
        if dist[i] == INF:
            print(f"{i} : INF")
        else:
            print(f"{i} : {dist[i]}")
```

2) C++

```cpp
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

const int INF = 1e9;  // 무한대를 나타내기 위한 값

vector<int> dijkstra(vector<vector<pair<int, int>>>& graph, int start, int n) {
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;  //최소힙

    vector<int> dist(n, INF);
    dist[start] = 0;
    pq.push({ 0, start });    //우선순위를 맨 앞에!

    while (!pq.empty()) {
```

```
            int current_dist = pq.top().first;
            int i = pq.top().second;
            pq.pop();

            if (current_dist > dist[i]) continue;       //이 코드를 추가하면 더 효율적

            for (auto& edge : graph[i]) {
                int j = edge.first;
                int w = edge.second;
                if (dist[i] + w < dist[j]) {
                    dist[j] = dist[i] + w;
                    pq.push({ dist[j], j });
                }
            }

        }
        return dist;
}

int main() {
        int n, m, st;
        cin >> n >> m >> st;

        vector<vector<pair<int, int>>> graph(n);
        vector<int> dist(n, INF);
        for (int i = 0; i < m; ++i) {
            int u, v, w;
            cin >> u >> v >> w;
            graph[u].push_back({ v, w });
        }


        vector<int> d = dijkstra(graph, st, n);

         for (int i = 0; i < n; ++i) {
            if (d[i] == INF) {
                cout <<  i << " : INF" << endl;
            }
            else {
                cout << i << " : " << d[i] << endl;
            }
        }

        return 0;
}
```

3) Java

```
import java.util.*;

public class Main {

    private static final int INF = Integer.MAX_VALUE;  // 무한대를 나타내기 위한 값

    public static int[] dijkstra(List<List<int[]>> graph, int start, int n) {
```

```
        PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[0]));
        int[] dist = new int[n];
        Arrays.fill(dist, INF);
        dist[start] = 0;
        pq.offer(new int[]{0, start});  // {거리, 정점}

        while (!pq.isEmpty()) {
            int[] current = pq.poll();
            int distance = current[0];
            int i = current[1];

            if (distance > dist[i]) continue;    //이 코드를 추가하면 더 효율적

            for (int[] edge : graph.get(i)) {
                int j = edge[0];
                int w = edge[1];
                if (dist[i] + w < dist[j]) {
                    dist[j] = dist[i] + w;
                    pq.offer(new int[]{dist[j], j});
                }
            }
        }
        return dist;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int m = scanner.nextInt();
        int st = scanner.nextInt();

        List<List<int[]>> graph = new ArrayList<>();
        for (int i = 0; i < n; ++i) {
            graph.add(new ArrayList<>());
        }

        for (int i = 0; i < m; ++i) {
            int u = scanner.nextInt();
            int v = scanner.nextInt();
            int w = scanner.nextInt();
            graph.get(u).add(new int[]{v, w});
        }

        int[] dist = dijkstra(graph, st, n);

        for (int i = 0; i < n; ++i) {
            if (dist[i] == INF) {
                System.out.println(i + " : INF");
            } else {
                System.out.println(i + " : " + dist[i]);
            }
        }

        scanner.close();
    }
}
```