6.4 최소 비용 Spanning tree

weighted, undirected graph의 spanning tree의 비용은 spanning tree의 edge weight의 합이다. minimum-cost spanning tree T는 edge weight의 합이 최소가 되는 spanning tree이다. minimum-cost spanning tree는 greedy-method로서 Kruskal, Prim, Sollin 알고리즘으로 구한다. greedy-method(탐욕적 방법)은 여러 경우 중에서 하나를 선택할 때마다 그 순간에 best라고 생각되는 것을 선택(매 순간마다 best만 선택하므로 탐욕적이라 부름)하는 방식이다. 매 순간마다 best를 선택한다고 해서 최종적인 선택이 best라고 항상 보장할 수는 없다. greedy-method가 잘 작동하는 문제는 greedy choice property와 optimal substructure 조건이 만족되는 경우이다. greedy choice property 조건은 앞의 best 선택이 이후의 best 선택에 영향을 주지 않는다는 것이다. optimal substructure 조건은 문제 전체에 대한 최적해(best 선택)가 부분 문제에 대해서도 역시 최적해라는 것이다. 이러한 두 가지 조건이 만족되지 않으면 greedy-method가 최적 해를 구하지 못한다.

greedy-method에서 각 stage별로 best 선택에 의한 optimal 해를 만든다. 초기 stage에서의 best 선택은 나중 stage에서 변경할 수 없다. minimum-cost spanning tree를 만들기 위한 best 선택은 least-cost를 선택할 때 사용하는 constraints는 다음과 같다: 1) 그래프의 edge 정보만 사용해야 한다. 2) n-1 edge만을 사용해야 한다. 3) cycle을 만드는 edge는 사용해서는 안 된다.

Minimal-cost spanning tree 문제는 자료구조로서 adjacency list와 set를 사용하여 알고리즘 구현하기에 좋은 예제이다. 어떠한 자료구조를 선택하여 재사용할 것인지에 대한 실습 과제로서 소스 코드를 완성해보는 실습을 수행한다. 정렬을 위해 min-heap을 사용하고 cycle 여부를 판단하기 위하여 set의 find()를 사용하도록 소스코드 6.2을 수정한다.

```
//소스 코드 6.2: Minimal Spanning Tree
//  minimal spanning tree:: Kruskal's source code
//  set 사용하여 MST 구현
import java.util.*;

class Edge implements Comparable<Edge> {
    int src, dest, weight;
    Edge(int s, int d, int w) {
        src = s; dest = d; weight = w;
    }

    @Override
    public int compareTo(Edge other) {
        return this.weight - other.weight; // 오름차순 정렬
    }
```

```
}

class Sets {
    private int[] parent;

    public Sets(int size) {
        parent = new int[size];
        Arrays.fill(parent, -1);
    }

    public int find(int i) {
        if (parent[i] < 0) return i;
        return parent[i] = find(parent[i]); // 경로 압축
    }

    public void union(int i, int j) {
        int root1 = find(i);
        int root2 = find(j);
        if (root1 == root2) return;

        int total = parent[root1] + parent[root2];
        if (parent[root1] > parent[root2]) {
            parent[root1] = root2;
            parent[root2] = total;
        } else {
            parent[root2] = root1;
            parent[root1] = total;
        }
    }

    public void display() {
        System.out.println("index = " + Arrays.toString(
            java.util.stream.IntStream.range(0, parent.length).toArray()));
        System.out.println("value = " + Arrays.toString(parent));
    }
}

class Graph {
    private int V;
    private List<Edge> edges = new ArrayList<>();
    private List<List<int[]>> adjList;

    public Graph(int v) {
        V = v;
        adjList = new ArrayList<>();
```

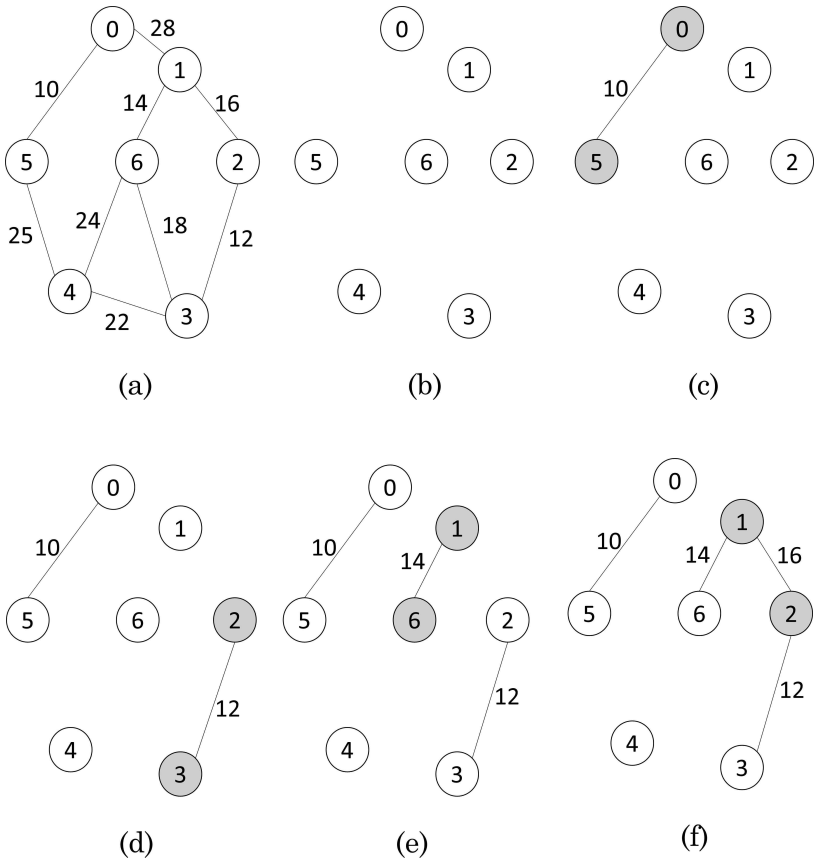
```
        for (int i = 0; i < V; i++) adjList.add(new LinkedList<>());
    }

    public void addEdge(int u, int v, int w) {
        edges.add(new Edge(u, v, w));
        adjList.get(u).add(new int[]{v, w});
        adjList.get(v).add(new int[]{u, w});
    }

    public void display() {
        for (int i = 0; i < V; i++) {
            System.out.print(i + " -> ");
            List<int[]> neighbors = adjList.get(i);
            if (neighbors.isEmpty()) {
                System.out.println("null");
                continue;
            }
            for (int[] edge : neighbors) {
                System.out.print(edge[0] + "(" + edge[1] + ") ");
            }
            System.out.println();
        }
    }

    public Graph kruskalMST() {
        Collections.sort(edges); // 가중치 기준 오름차순 정렬
        Sets sets = new Sets(V);
        Graph mst = new Graph(V);
        int edgeCount = 0;

        for (Edge e : edges) {
            int uRoot = sets.find(e.src);
            int vRoot = sets.find(e.dest);
            if (uRoot != vRoot) {
                mst.addEdge(e.src, e.dest, e.weight);
                sets.union(uRoot, vRoot);
                edgeCount++;
                if (edgeCount == V - 1) break;
            }
        }

        System.out.println("MST 생성 완료");
        return mst;
    }
}
```

```
public class MinimalSpanningTree {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("총 노드 수 입력: ");
        int n = sc.nextInt();

        Graph graph = new Graph(n);
        int select;

        do {
            System.out.println("\n1: 간선 추가, 2: 인접 리스트 출력, 3: 크루스칼 MST,
4: 종료");
            System.out.print("선택: ");
            select = sc.nextInt();

            switch (select) {
                case 1:
                    System.out.print("시작 노드: ");
                    int u = sc.nextInt();
                    System.out.print("도착 노드: ");
                    int v = sc.nextInt();
                    System.out.print("가중치: ");
                    int w = sc.nextInt();
                    graph.addEdge(u, v, w);
                    break;
                case 2:
                    graph.display();
                    break;
                case 3:
                    Graph mst = graph.kruskalMST();
                    System.out.println("MST 인접 리스트:");
                    mst.display();
                    break;
                case 4:
                    System.out.println("프로그램 종료");
                    break;
                default:
                    System.out.println("잘못된 입력");
            }
        } while (select != 4);

        sc.close();
    }
}
```

6.4.1 Kruskal 알고리즘

minimum-cost spanning tree을 만들기 위해 그림 6.7처럼 한 번에 하나의 edge만을 선택하여 set T에 추가한다[1]. 이를 위한 먼저 edge들의 weight를 오름차순으로 sort하고 edge가 cycle을 만들지 않으면 T에 추가한다. n vertices를 갖는 G에 대하여 n-1 edge가 포함되면 종료된다.

```
Algorithm Kruskal //[1]참조
begin
T = Ø
while ((T contains less than n−1 edges) && ( E not empty)) {
      choose an edge(v,w) from E of lowest cost;
      delete (v,w) from E;
      if ((v,w) does not create a cycle in T) add (v,w) to T;
      else discard (v,w);
}
if (T contains fewer than n−1 edges) cout <<"no spanning tree" <<endl;
end Kruskal
```



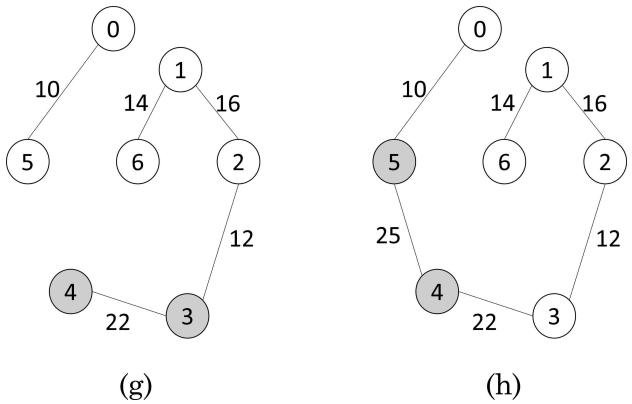


그림 6.7 Kruskal 알고리즘 적용 처리.

G = (V, E)에서 E의 lowest edge (v,w)를 선택하기 위하여 모든 edge를 minheap알고리즘으로 O(e log e)시간 복잡도로 sort한다. min heap은 next edge의 선택은 O(log e)으로 처리된다. (v,w)가 T에서 cycle을 만들지 않으면 T에 추가된다. T는 connected vertices로서 set으로 표현된다. v와 w가 T에서 connected이면 같은 set에 있게 된다.