자료구조설계

제 출 일	2017.10.29.
과 제 번 호	06
분 반	01
학 과	컴퓨터공학과
학 번	201602038
이 름	이 미 진

1. 구현 내용 설명

1) Edge

```
class Edge{
        int v; // start Vertex index
        int w; // end Vertex index
        int weight; // 가중치
        boolean selected; // 간선으로 선택되었는지 여부

        Edge(int v, int w, int weight){
                this.v = v;
                this.w = w;
                this.weight = weight;
        }
}
```

내부 클래스로 Edge를 만들어 start Vertex index, end Vertex index, 가중치, 간선으로 선택되었는지 여부를 판단하는 변수를 생성하였다.
생성자에서는 v,w,weight 만을 받는다. Edge 객체를 이용해 seleceted를 true라고 하면 간선으로 선택된 것이다.

2) public MST(int n)

```
 public MST(int n) throws IOException{
        this.size = n;
        this.parent = new int[size];
        Arrays.fill(parent, -1);//배열 -1로 초기화 (정점을 독립원소로 만듦)
        int v,w,weight;

        try{
                BufferedReader br = new BufferedReader(new
FileReader("C://input.txt"));

                String line = br.readLine();
                String[] split_line = line.split(" ");
                int size = Integer.parseInt(split_line[1]);

                input_edge = new Edge[size];    // 입력받은 edge를 저장하기 위한
배열 생성
                sort_edge = new Edge[size];     // sort 된 edge를 저장하기 위한
배열 생성
                Edge[] edge1 = new Edge[size]; // edge1 객체 생성

                while( i < size ) {
                        line = br.readLine();
                        split_line = line.split(" ");

                        v = Integer.parseInt(split_line[0]);
                        w = Integer.parseInt(split_line[1]);
                        weight = Integer.parseInt(split_line[2]);
                        Edge edge2 = new Edge(v, w, weight);  // 입력 받은
v,w,weight로 edge2 객체 생성
                        input_edge[i] = edge2; edge1[i] = edge2;

                        Edge edge3 = new Edge(edge1[i].v, edge1[i].w,
edge1[i].weight);

                                // 가중치 오름차순으로 정렬
                        int e;
                        for( e = i; 0 < e && edge1[i].weight < edge1[e-1].weight;
```

```
--e ){
                                        edge1[e] = edge1[e-1];
                                }
                                edge1[e] = edge3;
                                i++;
                        }

                                for( i = 0; i < size; i++ ) {
                                        sort_edge[i] = edge1[i];
                                }

                                br.close();
                        }
                        catch(FileNotFoundException e) {
                                e.printStackTrace();
                        }
                }
```

MST에서도 5주차 과제 Set과 비슷하게 try-catch와 파일 입력을 이용한다.
다른점은 Set에서는 Test 클래스에서만 파일을 입력받고, 입력받은 값을 Set으로 넘기는 것이었지만 MST에서는 Test와 MST 클래스 둘다 파일 입력을 받는다.
TestMST 클래스에선 파일의 길이를 읽어들여 MST에 size를 넘겨주기 위해 필요하다.
MST 클래스에선 파일을 입력받아 공백문자로 split한 후, split_line 배열에 저장한다. size엔 Integer.parseInt(split_line[1]) 한 값을 저장하고, 이를 이용해 입력받은 edge를 저장하기 위한 input_edge 배열, sort된 edge를 저장하기 위한 sort_edge 배열, edge1 객체를 생성한다.
size가 1보다 크면,위와 같이 입력받은 파일에서 split한 값을 split_line에 저장한다. 그리고 v에는 Integer.*parseInt*(split_line[0]), w에는 Integer.*parseInt*(split_line[1]), weight에는 Integer.*parseInt*(split_line[2]) 값을 각각 저장한다. 파일을 생성할 때, [start Vertex index] [end Vertex index] [weight] 이런식으로 입력했기 때문이다.
그 후에 값이 저장된 v,w,weight를 이용해 edge2 객체를 생성한다.
입력받은 edge가 저장되어 있는 input_edge 배열에 생성된 edge2 객체를 저장하고, 처음에 size로 생성한 edge1 객체에도 edge2를 저장한다.
edge3 객체에는 edge1 객체의 인덱스별로 v,w,weight를 각각 저장한다.
그 후에 edge1에 저장된 값들을 가중치를 이용해 오름차순으로 정렬한다.
정렬이 되면 edge1[e] 엔 edge3 값을 저장한다.
while 반복이 끝나면 sort_edge 배열에 sort된 원소들이 저장된 edge1 배열을 저장한다.

3) public void weightedUnion(int I, int j)

```
        public void weightedUnion(int i, int j){
                // i가 속한 집합과 j가 속한 집합을 합집합으로 만든다
                int i_2 = i;        // 현재 i 를 i_2 변수에 저장
                int j_2 = j;        // 현재 j 를 j_2 변수에 저장
                int parent_i = parent[i]; // parent[i]의 값을 parent_i에 저장
                int parent_j = parent[j]; // parent[j]의 값을 parent_j에 저장


                for( ; parent[i] >= 0; ) {// root 노드일 경우까지
                        i = parent[i];
                }
                parent_i = i;
                for( ; parent[j] >= 0; ) {// root 노드일 경우까지
                        j = parent[j];
                }
                parent_j = j;

                if( parent[parent_i] < parent[parent_j] ) {         // i 집합이 j
집합보다 작을 경우
                        parent[parent_i] += parent[parent_j]; // i집합 = i 집합 + j
집합
                        parent[parent_j] = parent_i;
                        parent_j = parent_i;

                }
                else {   // i 집합이 j 집합보다 클 경우
                        parent[parent_j] += parent[parent_i]; // j 집합 = j 집합 +
i 집합
                        parent[parent_i] = parent_j;
                        parent_i = parent_j;
                }

                while( parent[i_2] > parent_i && parent[j_2] > parent_j ) {
// root 노드일 경우까지
                        int tmp;
                        if( parent[i_2] == parent_i ) {
                                tmp = parent[i_2];
                                parent[tmp] = collapsingFind(tmp);        //
parent를 tmp로 바꿈
                        }
                        if( parent[j_2] == parent_j ) {
                                tmp = parent[j_2];
                                parent[tmp] = collapsingFind(tmp);        //
parent를 tmp로 바꿈
                        }
                }
        }
```

weightedUnion(int I,int j) 메소드는 5주차 과제와 동일한 코드를 사용하였다.
weightedUnion(int I,int j) 메소드는 I가 속한 집합과 j가 속한 집합을
합집합으로 만들어주는 메소드이다. I와 j의 초기값을 각각 I_2와 j_2에
저장하고, parent[i],parent[j]를 각각 parent_i,parent_j 변수에 저장한다.
우선, weightedUnion(i,j) 형식으로 I와 j를 출력해주어, 어떤 집합에 대한 것인지 명시하도록 하였다.
for문을 통해 parent[i] parent[j]가 root 노드일 경우까지 반복하면서
parent[i]는 I에 저장, parent[j]는 j에 저장한다.
반복이 끝나면 parent_i에 I를, parent_j에 j를 저장해둔다.
I와 j로 값이 바뀐 parent_i와 parent_j를 통해 합집합을 만드는 과정을 수행한다.
parent[parent_i] (i 집합) 이 parent[parent_j] (j 집합)보다 작을 경우,
parent[parent_i] (i 집합) 에 parent[parent_j] (j 집합)을 더해 parent[parent_i] 값에 저장한다.
그럼 I 집합에는 I집합 + j집합을 한 값이 들어가게 된다.

그 후에 root를 재설정하는 과정을 거친다.
반대의 경우도 I와 j만 서로 변경하여 수행한다.
합집합으로 만드는 과정이 끝나면 parent[i_2] 와 parent[j_2]가 root 노드일 경우와, parent[i_2] 가 parent_i와 같을 경우에 tmp에 parent[i_2]를 저장하고 collapsingFind(tmp)를 호출해 parent[tmp]에 저장한다.
j_2의 경우에도 동일한 과정을 거친다.

4) public int collapsingFind(int I)

```
        public int collapsingFind(int i) {
                // i가 속한 집합의 root를 찾는다.
                for( ; parent[i] > 0 ;) {
                        i = parent[i];    // 자식에서 부모로 이동
                }
                return i;         // i = root 노드
        }
```

collpasingFind() 메소드는 5주차 과제와 동일한 코드를 사용하였다.
collapsingFind() 메소드는 I가 속한 집합의 root를 찾는 메소드이다.
parent[i]가 root 노드일 경우까지 반복하면서 I에 parent[i]를 저장하여 자식에서 부모로 이동하는 과정을 수행한다.
I를 return 해주면서 I가 root 노드가 된다.

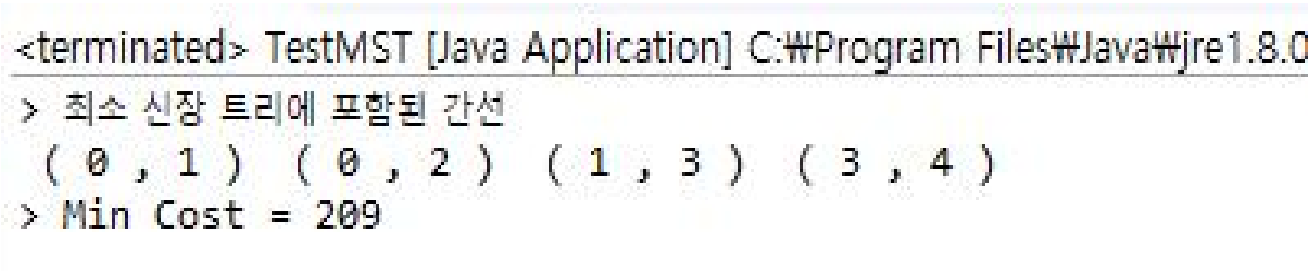
5) public void Kruskal( )

```
public void Kruskal(){
        int connected_edges = 0;        // 연결된 선분 수
        int check_edges = 0;    // 확인된 edge수
        int root_v = 0; // v의 root
        int root_w = 0; // w의 root
        int i = 0, j = 0;
        int length = input_edge.length;
        for( ; connected_edges != size && check_edges < length ; ) {
                root_v = sort_edge[check_edges].v;
                root_w = sort_edge[check_edges].w;

                if( parent[root_v] != 0 ){
                        for( ; parent[root_v] >= 0; ) {   // root 노드 탐색
                        root_v = parent[root_v];
                        }
                }
                if( parent[root_w] != 0 ) {
                        for( ; parent[root_w] >= 0 ; ) { // root 노드 탐색
                                root_w = parent[root_w];
                        }
                }

                if( root_v != root_w ){  // root가 서로 다르면 weightedUnion 호출
                        weightedUnion(sort_edge[check_edges].v,
sort_edge[check_edges].w);
                        sort_edge[check_edges].selected = true; // 간선으로
선택되었으므로 true
                        connected_edges++;      // 연결 된 선분 수 증가
                }
                        check_edges++; // 다음 edge 탐색
                }

                // sort, weightedUnion,collapsingFind 등 연산을 한 결과를
input_edge에 저장
```

```
                while( i < length   && j < length ) {
                        if( input_edge[i].v == sort_edge[j].v ) {
                                if( input_edge[i].w == sort_edge[j].w ) {
                                        input_edge[i] = sort_edge[j];
                                        j++;
                                }
                                i++;
                        }
                }

                System.out.println("> 최소 신장 트리에 포함된 간선");
                System.out.print(">");
                for( i=0; i < length; i++ ){
                        if( input_edge[i].selected == true ){
                                System.out.print(" ( "+ input_edge[i].v +" , "+
input_edge[i].w +" ) ");
                                minCost += input_edge[i].weight;
                        }       // 연결되어 있으면 최소 신장 트리에 포함된 간선을
출력, minCost와 가중치를 더한다.
                }
                System.out.println();
                System.out.println("> Min Cost = "+ minCost);    // minCost 출력
        }
}
```

Kruskal() 알고리즘은 최소 신장 트리를 형성하기 위한 알고리즘이다.
연결된 선분 수를 저장할 connected_edges 변수, 확인된 edge 수를 저장할 check_edges 변수, v의 root를 저장할 root_v, w의 root를 저장할 root_w 변수를 선언하였다.
connected_edges가 size 와 다르고, check_edges가 input_edge의 길이보다 작을 때, check_edges를 증가시키면서 반복을 수행한다.
root_v에는 sort_edge에 check_edges 인덱스가 들어가고, 그 값의 v를 저장한다. root_w에는 sort_edge에 check_edges 인덱스가 들어가고, 그 값의 w를 저장한다.
만약 parent[root_v]가 0이 아니라면, root노드를 탐색해 root_v에 parent[root_v]를 저장한다. w의 경우도 동일한 과정을 거친다.
그 후에 root_v와 root_w가 같지 않으면, weightedUnion 메소드를 호출하여 합집합을 만든다.
그러면 sort_edge[check_edges]는 간선으로 선택되었으므로 selected가 true가 된다. 그리고 connected_edges를 증가시켜줘야 한다.
for 반복이 끝나면, length가 I(0),j(0)보다 클 때의 반복을 해준다.
만약 input_edge[i]의 v가 sort_edge[j]의 v와 같으면, input_edge[i]의 w와 sort_edge[j]의 w가 같은지도 검사하고, 두 조건이 맞으면 input_edge[i]에는 sort_edge[j]값을 저장한다. 각 조건 검사를 끝낼 때마다, I와 j를 각각 증가시켜준다.
최소 신장 트리에 포함된 간선을 출력할 때에는, input_edge.length만큼 인덱스를 주고, input_edge[i]의 selected가 true일 경우에만 간선을 출력해준다. (0 , 1) (0 , 2) (1 , 3) (3 , 4) 이런식으로 출력이 된다. 그리고 selected가 true 이므로, minCost에도 input_edge[i]의 가중치를 더해주고 저장한다.
반복이 끝나면 minCost도 함께 출력한다.

2. 실행 결과 화면


<terminated> TestMST [Java Application] C:₩Program Files₩Java₩jre1.8.0
> 최소 신장 트리에 포함된 간선
(0 , 1) (0 , 2) (1 , 3) (3 , 4)
> Min Cost = 209