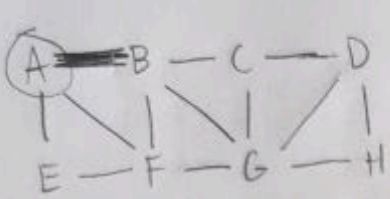
Assignment 13

1) Prim's Algorithm

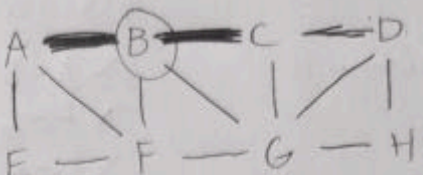
goes node by node, starting at a node

1.



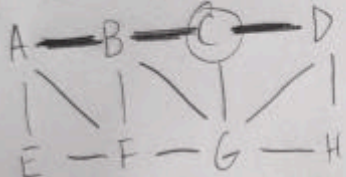
~~A - B 1~~
A - E 4
A - F 8

2.



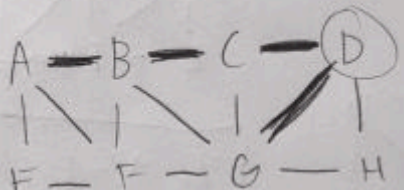
~~B - C 2~~
B - F 4
A - E 4
B - G 6
A - F 8

3.



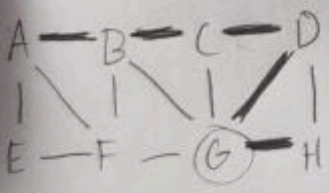
~~C - D 3~~
B - F 4
A - E 4
C - G 6
B - G 6
A - F 8

4.



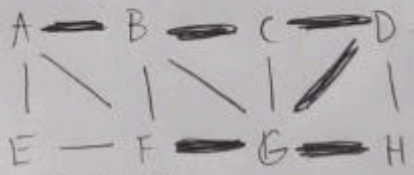
~~D - G 1~~
D - H 4
B - F 4
A - E 4
C - G 6
B - G 6
A - F 8

5.



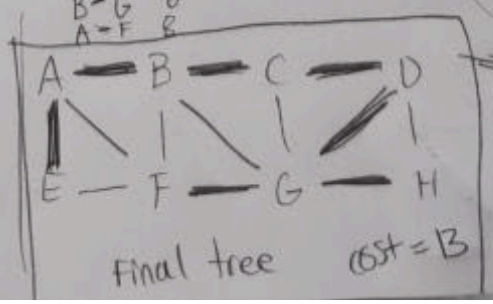
~~G - H 1~~
F - G 1
D - H 4
B - F 4
A - E 4
C - G 6
B - G 6
A - F 8

6.



~~F - G 1~~
D - H 4
B - F 4
A - E 4
F - E 5
C - G 6
B - G 6
A - F 8

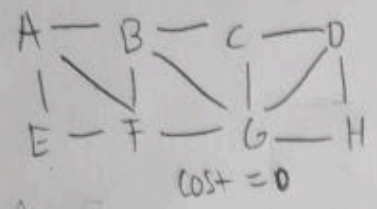
7.



skips nodes already in tree

D - H 4
B - F 4
~~A - E 4~~
F - E 5
C - G 6
B - G 6
A - F 8

2) Kruskall's Algorithm takes all possible edges then sorts indiscriminately

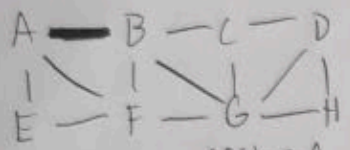


cost = 0

PQ consists of:

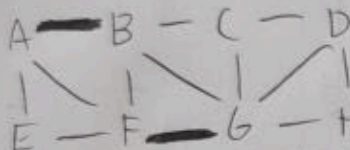
A-B	1	B-C	2	D-H	4
F-G	1	C-D	3	E-F	5
G-H	1	A-E	4	C-G	6
G-D	1	B-F	4	B-G	6
				A-F	8

1. A-B



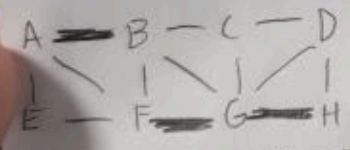
cost = 1

2. F-G



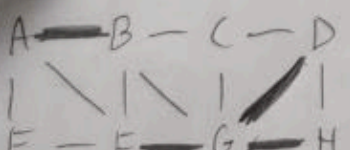
cost = 2

3. G-H



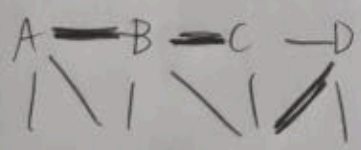
cost = 3

4. G-D



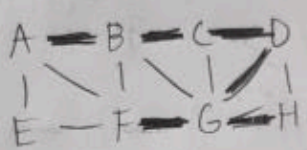
cost = 4

5. B-C



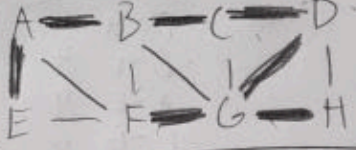
cost = 6

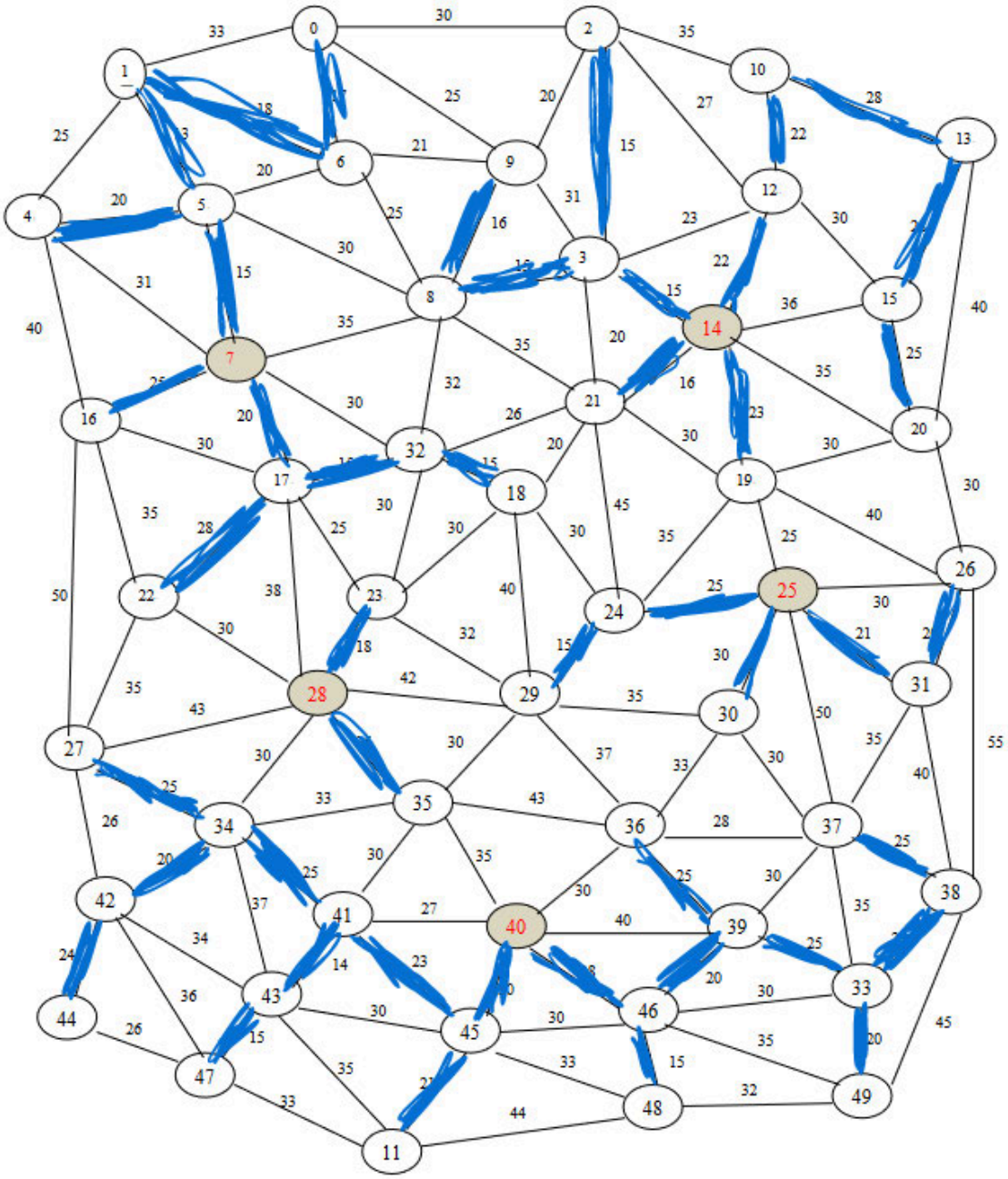
6. C-D



cost = 9

7. A-E cost = 13



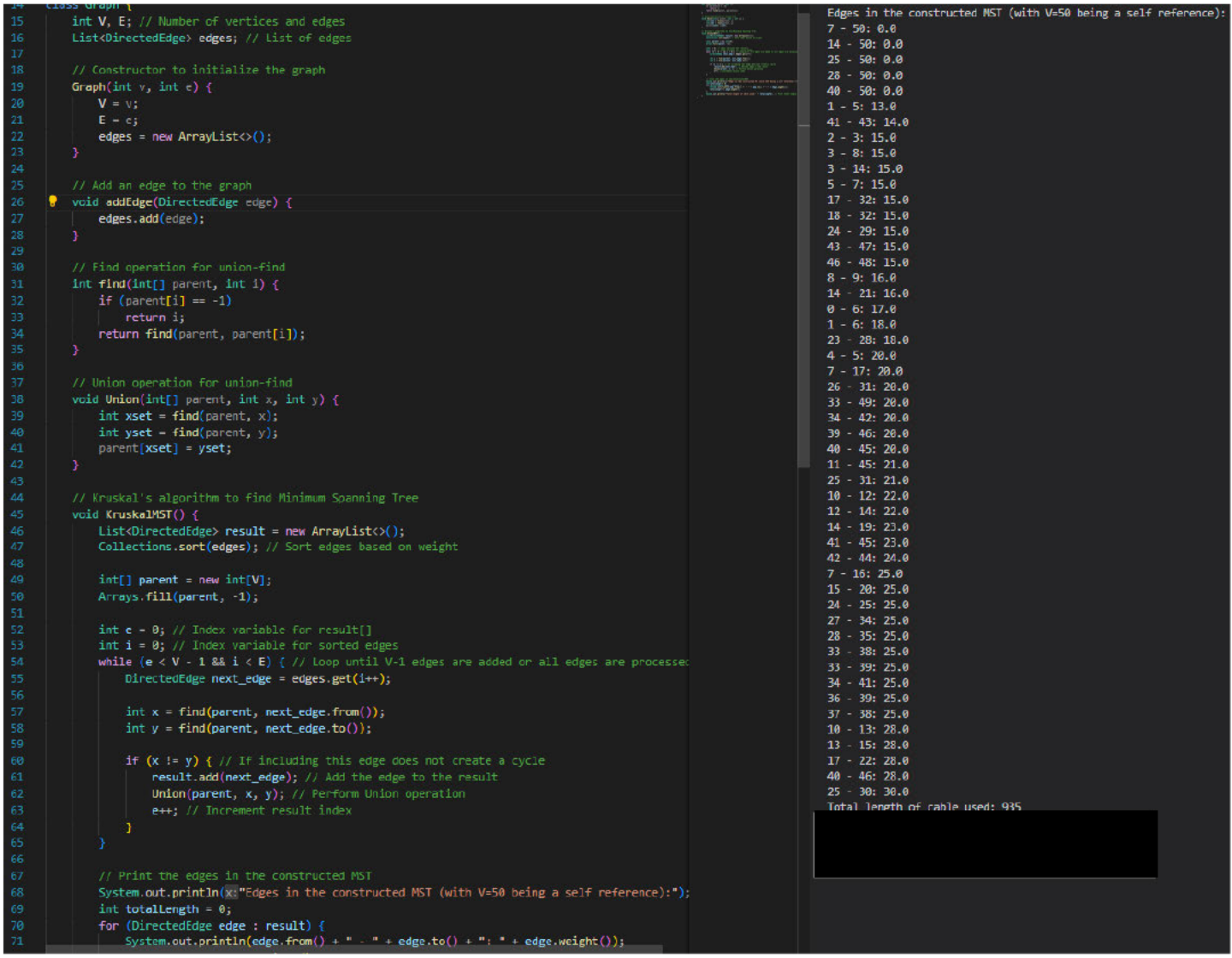


TOTAL COST IS 935

For the shortest node:

I used my code with data centering each field station. I found that **station 14 has the best cost, with 987.**

Coding output (though it is included in code):



```
    int V, E; // Number of vertices and edges
    List<DirectedEdge> edges; // List of edges

    // Constructor to initialize the graph
    Graph(int v, int e) {
        V = v;
        E = e;
        edges = new ArrayList<>();
    }

    // Add an edge to the graph
    void addEdge(DirectedEdge edge) {
        edges.add(edge);
    }

    // Find operation for union-find
    int find(int[] parent, int i) {
        if (parent[i] == -1)
            return i;
        return find(parent, parent[i]);
    }

    // Union operation for union-find
    void Union(int[] parent, int x, int y) {
        int xset = find(parent, x);
        int yset = find(parent, y);
        parent[xset] = yset;
    }

    // Kruskal's algorithm to find Minimum Spanning Tree
    void KruskalMST() {
        List<DirectedEdge> result = new ArrayList<>();
        Collections.sort(edges); // Sort edges based on weight

        int[] parent = new int[V];
        Arrays.fill(parent, -1);

        int e = 0; // Index variable for result[]
        int i = 0; // Index variable for sorted edges
        while (e < V - 1 && i < E) { // Loop until V-1 edges are added or all edges are processe
            DirectedEdge next_edge = edges.get(i++);

            int x = find(parent, next_edge.from());
            int y = find(parent, next_edge.to());

            if (x != y) { // If including this edge does not create a cycle
                result.add(next_edge); // Add the edge to the result
                Union(parent, x, y); // Perform Union operation
                e++; // Increment result index
            }
        }

        // Print the edges in the constructed MST
        System.out.println("Edges in the constructed MST (with V=50 being a self reference):");
        int totalLength = 0;
        for (DirectedEdge edge : result) {
            System.out.println(edge.from() + " - " + edge.to() + ": " + edge.weight());
```

```
Edges in the constructed MST (with V=50 being a self reference):
7 - 50: 0.0
14 - 50: 0.0
25 - 50: 0.0
28 - 50: 0.0
40 - 50: 0.0
1 - 5: 13.0
41 - 43: 14.0
2 - 3: 15.0
3 - 8: 15.0
3 - 14: 15.0
5 - 7: 15.0
17 - 32: 15.0
18 - 32: 15.0
24 - 29: 15.0
43 - 47: 15.0
46 - 48: 15.0
8 - 9: 16.0
14 - 21: 16.0
0 - 6: 17.0
1 - 6: 18.0
23 - 28: 18.0
4 - 5: 20.0
7 - 17: 20.0
26 - 31: 20.0
33 - 49: 20.0
34 - 42: 20.0
39 - 46: 20.0
40 - 45: 20.0
11 - 45: 21.0
25 - 31: 21.0
10 - 12: 22.0
12 - 14: 22.0
14 - 19: 23.0
41 - 45: 23.0
42 - 44: 24.0
7 - 16: 25.0
15 - 20: 25.0
24 - 25: 25.0
27 - 34: 25.0
28 - 35: 25.0
33 - 38: 25.0
33 - 39: 25.0
34 - 41: 25.0
36 - 39: 25.0
37 - 38: 25.0
10 - 13: 28.0
13 - 15: 28.0
17 - 22: 28.0
40 - 46: 28.0
25 - 30: 30.0
Total length of cable used: 935
```