CX2101 Algorithm Design and Analysis

Tutorial 2 (Graphs)

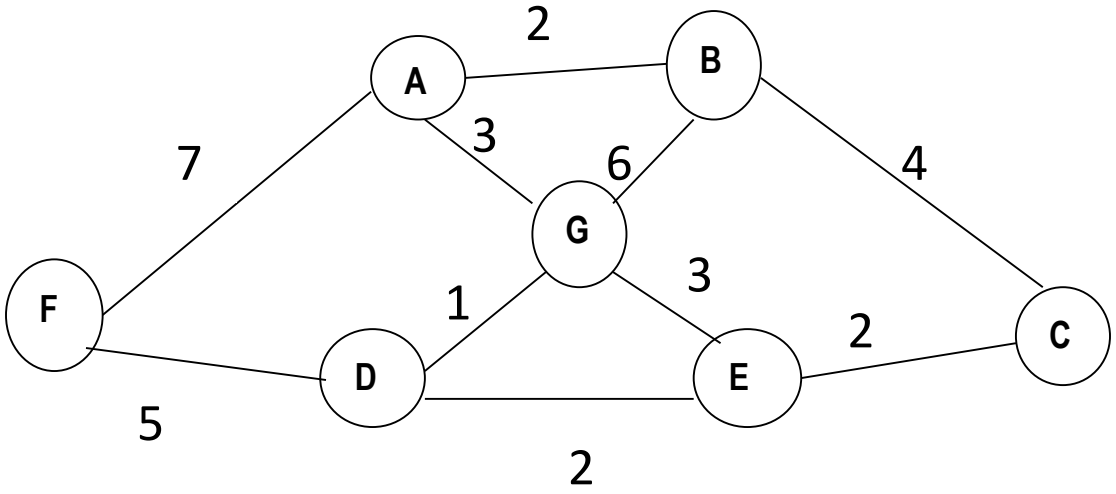
Week 8: Q7-Q9

This Tutorial

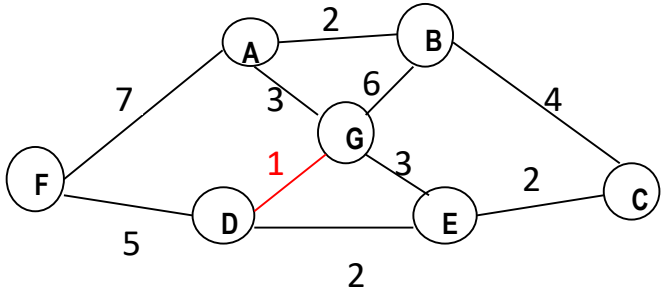
- Minimum spanning tree algorithm – Kruskal's
- Union-find data structure

Question 7

- Execute by hand the Kruskal's algorithm (with the weighted QuickUnion algorithm for Union-Find) for finding minimum spanning tree (MST) on the graph below. Show the contents of arrays *id* and *sz* at each step when an edge is added to the MST.

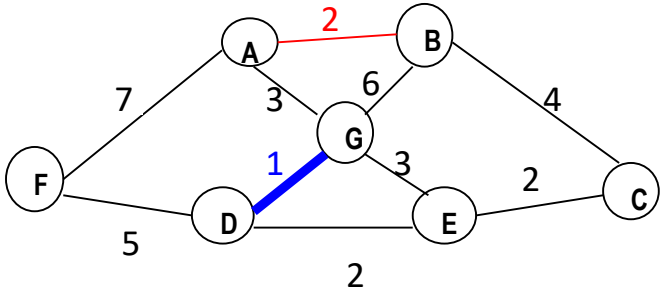


After initialization:

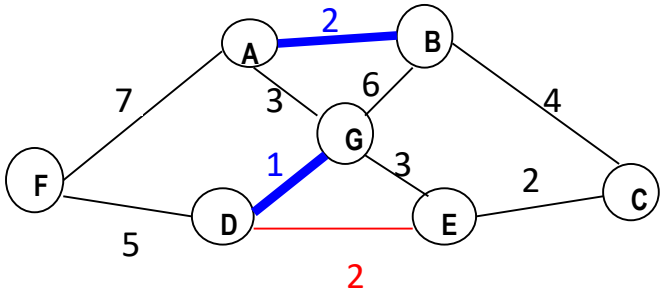


	A	B	C	D	E	F	G
id	A	B	C	D	E	F	G
sz	**1**	**1**	**1**	**1**	**1**	**1**	**1**

1st iteration:

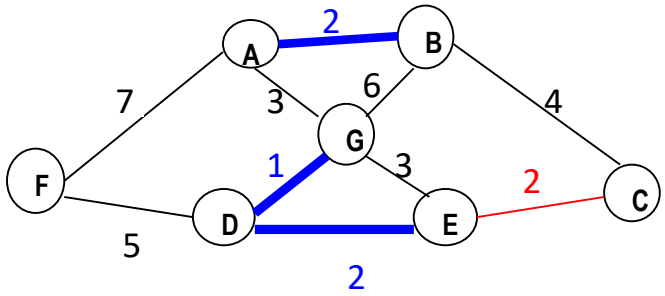


	A	B	C	D	E	F	G
id	A	B	C	D	E	F	D
sz	**1**	**1**	**1**	**2**	**1**	**1**	**1**



2nd iteration:

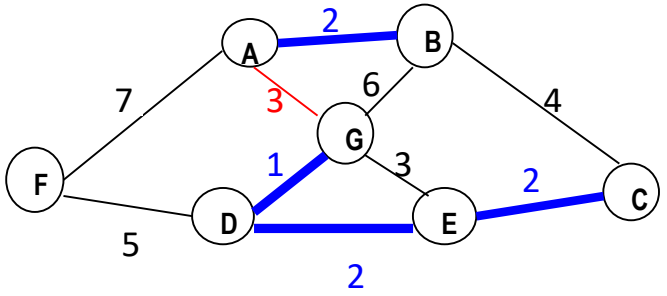
	A	B	C	D	E	F	G
id	A	A	C	D	E	F	D
sz	**2**	**1**	**1**	**2**	**1**	**1**	**1**



3rd iteration:

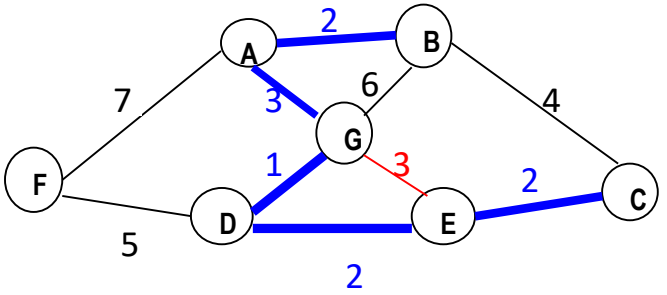
	A	B	C	D	E	F	G
id	A	A	C	D	D	F	D
sz	**2**	**1**	**1**	**3**	**1**	**1**	**1**

4th iteration:



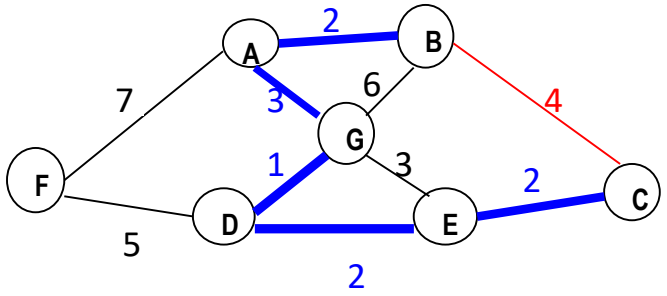
	A	B	C	D	E	F	G
id	A	A	D	D	D	F	D
sz	**2**	**1**	**1**	**4**	**1**	**1**	**1**

5th iteration:



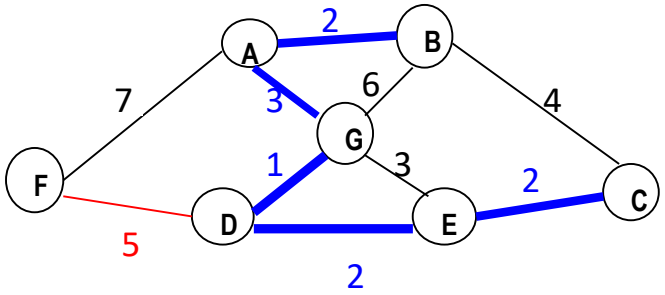
	A	B	C	D	E	F	G
id	D	A	D	D	D	F	D
sz	**2**	**1**	**1**	**6**	**1**	**1**	**1**

6th iteration:



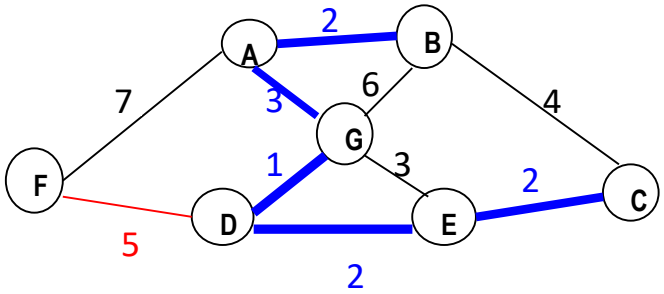
	A	B	C	D	E	F	G
id	D	A	D	D	D	F	D
sz	**2**	**1**	**1**	**6**	**1**	**1**	**1**

7th iteration:

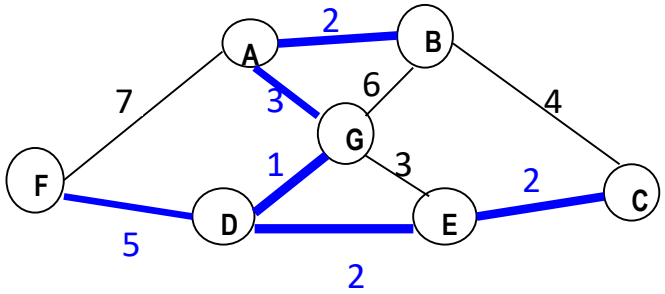


	A	B	C	D	E	F	G
id	D	A	D	D	D	F	D
sz	**2**	**1**	**1**	**6**	**1**	**1**	**1**

8th iteration:



	A	B	C	D	E	F	G
id	D	A	D	D	D	D	D
sz	**2**	**1**	**1**	**7**	**1**	**1**	**1**



Question 8

- If the input graph to the Kruskal's algorithm is given in an adjacency matrix, what is the time complexity of the algorithm?

```
public class KruskalMST
{
   private Queue<Edge> mst = new Queue<Edge>();

   public KruskalMST(EdgeWeightedGraph G)
   {
      MinPQ<Edge> pq = new MinPQ<Edge>(G.edges());

      UF uf = new UF(G.V());
      while (!pq.isEmpty() && mst.size() < G.V()-1)
      {
         Edge e = pq.delMin();
         int v = e.either(), w = e.other(v);
         if (!uf.connected(v, w))
         {
            uf.union(v, w);
            mst.enqueue(e);
         }
      }
   }

   public Iterable<Edge> edges()
   {  return mst;  }
}
```

build priority queue (or sort) — O(|E|) — O(|V|²)

O(|V|)

greedily add edges to MST — O(|E| log|E|)

edge v–w does not create cycle — O(|E| log*|V|)

merge sets — O(|V| log*|V|)

add edge to MST — O(|V|)

Overall: O(|E| log|E|) — **O(|E| log|E| + |V|²)**

Question 9

- Design an algorithm to check whether a given undirected graph $G = (V, E)$ contains a cycle or not. Analyze the complexity of the algorithm in terms of $|V|$ and $|E|$.
- Ans: One solution is to use the union-find.
  - Process every edge (u, v) one by one
  - If u and v are from different components, then union(u, v)
  - Otherwise, return TRUE
  - After all edges are processed, return FALSE
- Complexity: O(|V| + |E|log*|V|)

What we have exercised

- Minimum spanning tree algorithm
  - Kruskal's algorithm
    - Running of the algorithm
    - Data structure contents in each iteration
    - Its complexity depends on the implementation
- Union-find data structure
  - Can be used to design an algorithm for checking if a graph contains a cycle