CX2101 Algorithm Design and Analysis

Tutorial 2: Graphs

School of Computer Science and Engineering *Nanyang Technological University*

Week 6 (Q1-Q3):

Q1 Apply the Dijkstra's algorithm on the graph represented by the following adjacency matrix to find the shortest distances and the shortest paths from vertex 1 to the other vertices. Show the contents of arrays S, d and pi after each iteration of the while loop.

vertex	1	2	3	4	5
1	0	4	2	6	8
2	∞	0	∞	4	3
3	∞	∞	0	1	∞
4	∞	1	∞	0	3
5	∞	∞	∞	∞	0

Q2 Let $G = (V, E, W)$ be a weighted graph, and let s and z be distinct vertices. In the graph, there may be more than one shortest path from s to z. Explain how to modify Dijkstra's shortest-path algorithm to determine the number of distinct shortest paths from s to z. Assume all edge weights are positive.

Q3 Dijkstra's algorithm requires that the input graph has all edges being non-negative. Give an example where Dijkstra's algorithm does not work correctly with negative weights.

Week 7 (Q4-Q6):

Q4 Execute by hand the Prim's algorithm for finding minimum spanning tree (MST) on the graph in Figure 2.1, starting from vertex G. Show the contents of arrays S, d and pi after each iteration of the while loop when a vertex is added to the MST.

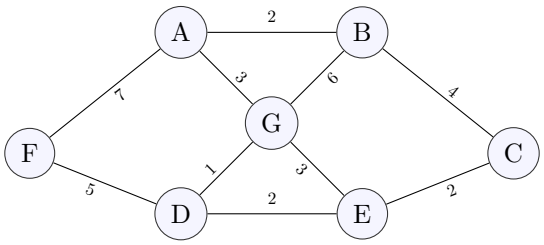


Figure 2.1: Graph for Q4

Q5 In a weighted undirected graph, is the path between two vertices in a minimum spanning tree always the shortest path (i.e. a path with the minimum weight) between the two vertices in the graph? If your answer is yes, give a proof; otherwise, give a counterexample.

Q6 Draw a connected graph with five nodes, six edges of respective weights 5, 6, 7, 8, 9, 10, and a minimum spanning tree of weight 28. Is it possible to have an MST of weight 29? If yes, draw the graph; otherwise, provide your justification.

Week 8 (Q7-Q9):

Q7 Execute by hand the Kruskal's algorithm (with the weighted QuickUnion algorithm for Union-Find) for finding minimum spanning tree (MST) on the graph in Figure 2.1. Show the contents of arrays id and sz at each step when an edge is added to the MST.

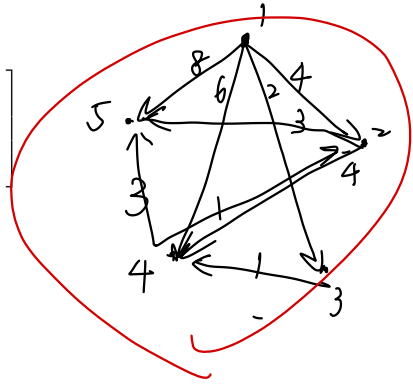
Q8 If the input graph to the Kruskal's algorithm is given in an adjacency matrix, what is the time complexity of the algorithm?

Q9 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 $|E|$.

Week 6

Q1 Apply the Dijkstra's algorithm on the graph represented by the following adjacency matrix to find the shortest distances and the shortest paths from vertex 1 to the other vertices. Show the contents of arrays S, d and pi after each iteration of the while loop.

vertex	1	2	3	4	5
1	0	4	2	6	8
2	∞	0	∞	4	3
3	∞	∞	0	1	∞
4	∞	1	∞	0	3
5	∞	∞	∞	∞	0



图不好清楚!

After initialization:

	1	2	3	4	5
S	0	0	0	0	0
d	0	∞	∞	∞	∞
pi	null	null	null	null	null

iter1

	1	2	3	4	5
S	1	0	0	0	0
d	0	4	2	6	8
pi	null	1	1	1	1

iter2

	1	2	3	4	5
S	1	0	1	0	0
d	0	4	2	~~3~~ 6	~~8~~
pi	null	1	1	~~1~~ 3	~~1~~

iter3 (no updates)

	1	2	3	4	5
S	1	~~0~~	1	~~0~~ 1	0
d	0	4	2	~~6~~ 3	~~6~~ 4
pi	null	1	1	~~3~~ 4	~~4~~

iter4

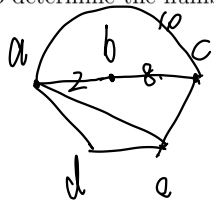
	1	2	3	4	5
S	1	1	1	~~1~~ 0	~~0~~
d	0	4	2	~~3~~ 3	~~6~~
pi	null	1	1	~~1~~ 3	~~3~~ 4

iter5

	1	2	3	4	5
S	1	1	1	1	1
d	0	4	2	~~3~~ 3	~~6~~
pi	null	1	1	~~1~~ 3	~~3~~ 4

Shortest paths:
1 – 2
1 – 3
1 – 3 – 4
1 – 3 – 4 – 5

Q2 Let $G = (V, E, W)$ be a weighted graph, and let s and z be distinct vertices. In the graph, there may be more than one shortest path from s to z. Explain how to modify Dijkstra's shortest-path algorithm to determine the number of distinct shortest paths from s to z. Assume all edge weights are positive.

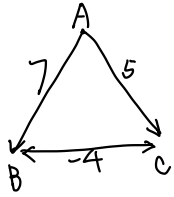


Update 时遇到 distance 相同的情况.

Include a new array ⇒ # of shortest path. initialized to 1

1. when vertex included, # of shortest + 1.
2. when try to update but fail (same distance) # of shortest path +1

Q3 Dijkstra's algorithm requires that the input graph has all edges being non-negative. Give an example where Dijkstra's algorithm does not work correctly with negative weights.



Take A to be the src.

iter 1

	A	B	C
S	1	0	0
d	0	7	5
pi	A	A	A

iter 2

	A	B	C
S	1	0	1
d	0	1	5
pi	A	C	A

iter 3

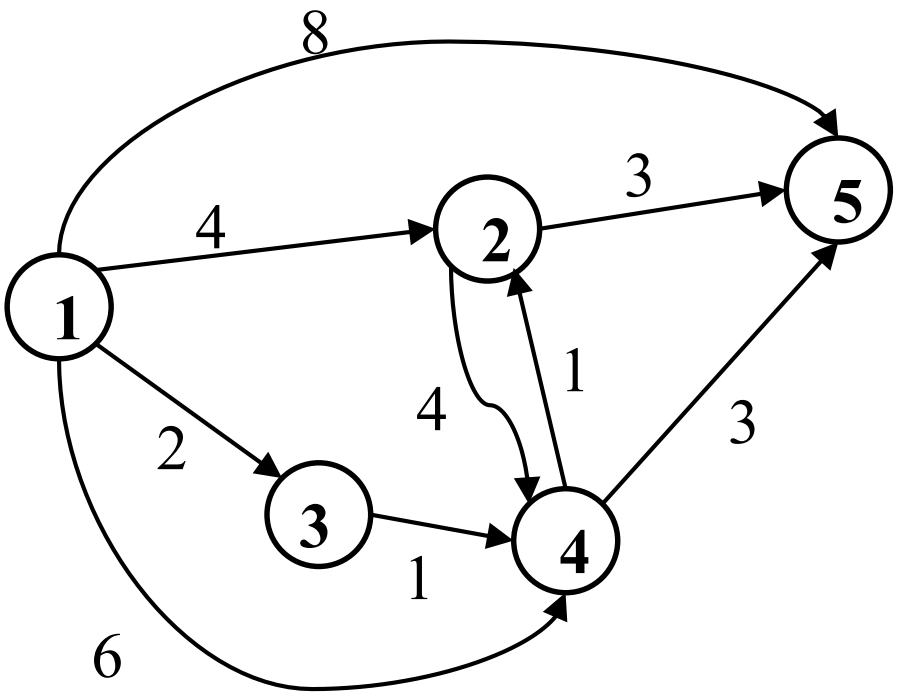
	A	B	C
s	1	1	1
d	0	1	1
pi	A	C	B

Question 1

Apply the Dijkstra's algorithm on the graph represented by the following adjacency matrix to find the shortest distances and the shortest paths from vertex 1 to the other vertices. Show the contents of arrays S, d and pi after each iteration of the while loop.

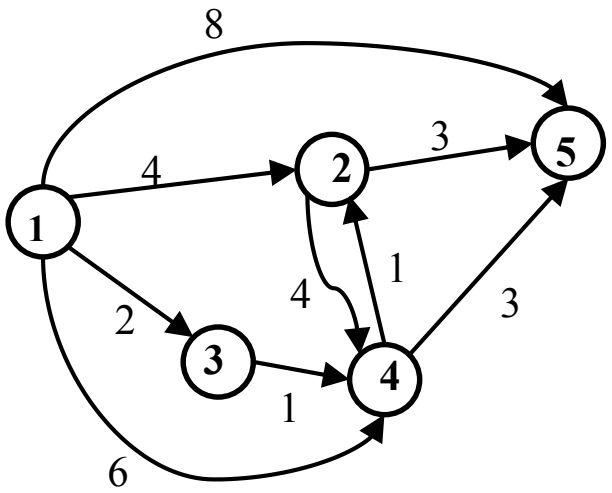
vertex	1	2	3	4	5
1	0	4	2	6	8
2	∞	0	∞	4	3
3	∞	∞	0	1	∞
4	∞	1	∞	0	3
5	∞	∞	∞	∞	0

vertex	1	2	3	4	5
1	0	4	2	6	8
2	∞	0	∞	4	3
3	∞	∞	0	1	∞
4	∞	1	∞	0	3
5	∞	∞	∞	∞	0



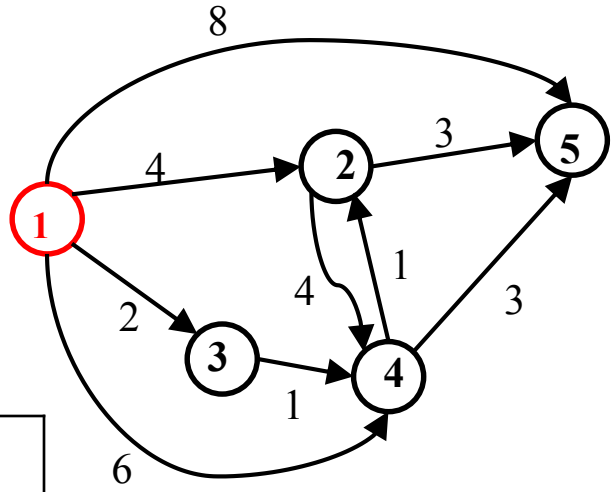
After initialization:

	1	2	3	4	5
S	0	0	0	0	0
d	0	∞	∞	∞	∞
pi	null	null	null	null	null



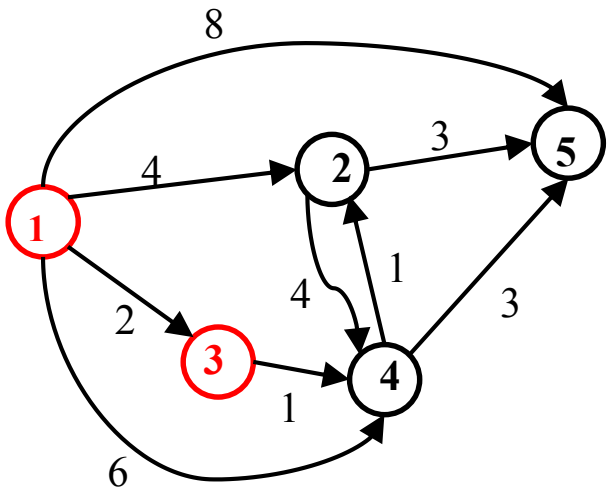
First iteration:

	1	2	3	4	5
S	1	0	0	0	0
d	0	4	2	6	8
pi	null	1	1	1	1



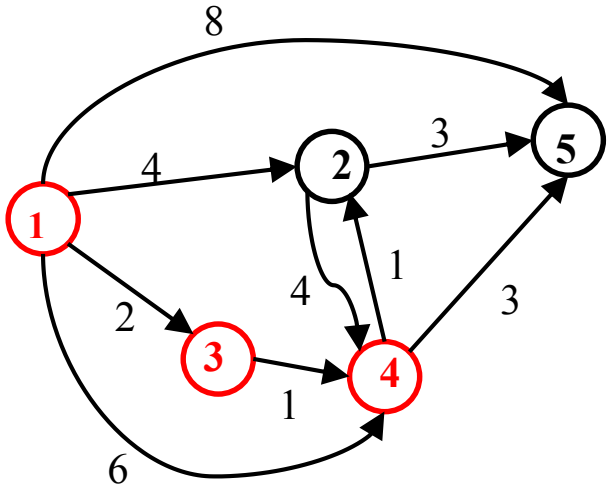
2nd iteration:

	1	2	3	4	5
S	1	0	1	0	0
d	0	4	2	3	8
pi	null	1	1	3	1



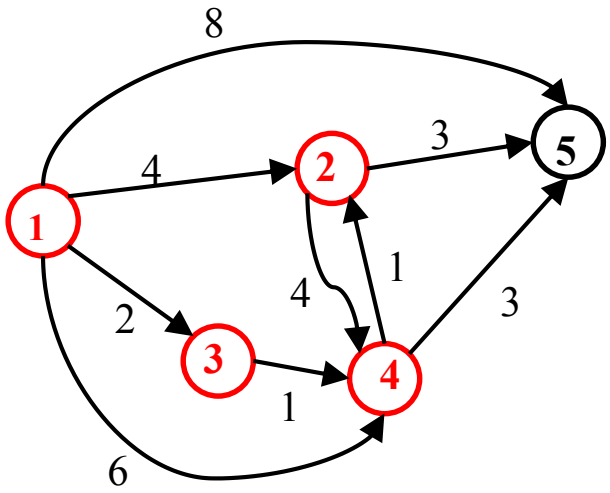
3rd iteration:

	1	2	3	4	5
S	1	0	1	1	0
d	0	4	2	3	6
pi	null	1	1	3	4



4th iteration:

	1	2	3	4	5
S	1	1	1	1	0
d	0	4	2	3	6
pi	null	1	1	3	4



5th iteration:

	1	2	3	4	5
S	1	1	1	1	1
d	0	4	2	3	6
pi	null	1	1	3	4

Shortest paths:

1 – 2

1 – 3

1 – 3 – 4

1 – 3 – 4 – 5

Question 2

- Let $G = (V, E, W)$ be a weighted graph, and let s and z be distinct vertices. In the graph, there may be more than one shortest path from s to z. Explain how to modify Dijkstra's shortest-path algorithm to determine the number of distinct shortest paths from s to z. Assume all edge weights are positive.

Question 2

```
shortest_paths( Graph g, Node source )
// The array count[] records the number of
// shortest paths from
// source to each of the vertices
{
     for each vertex v {
            d[v] = infinity;
            pi[v] = null pointer;
            S[v] = 0;
            count[v] = 0;
     }
     d[source] = 0;
     count[source] = 1;
     put all vertices in queue, Q, in d[v]'s order;
```

```
while not Empty(Q) {
    u = ExtractCheapest( Q );
    S[u] = 1; /* Add u to S */
    for each vertex v adjacent to u
        if (S[v] != 1 && d[v] > d[u] + w[u,v]) {
            remove v from Q;
            d[v] = d[u] + w[u,v];
            pi[v] = u;
            // the shortest paths to u extends to v, the
            // previously known shortest path(s) to v is replaced
            count[v] = count[u];
            insert v into Q according to its d[v];
        }
        else if (d[v] == d[u] + w[u, v]) {
            // additional shortest path(s) through u to v is found
            count[v] += count[u];
        }
    }
}
```

Question 3

- Dijkstra's algorithm requires that the input graph has all edges being non-negative. Give an example where Dijkstra's algorithm does not work correctly with negative weights.

Question 3

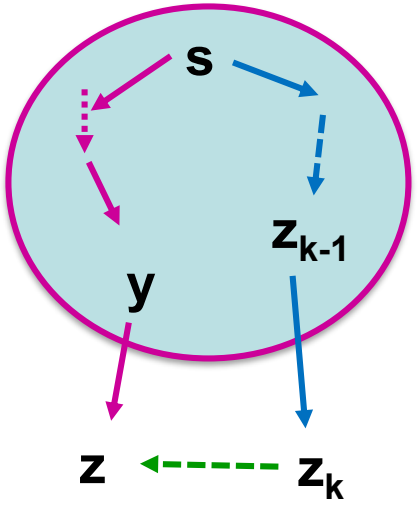
- Thoughts: Assumption of "non-negative weights" is used in the proof of Theorem D1.

$W(P) = d[y] + W(y, z)$

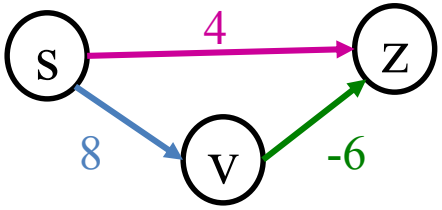
$W(P') = d[z_{k-1}] + W(z_{k-1}, z_k)$ + distance from z_k to z

Note that: $d[z_{k-1}] + W(z_{k-1}, z_k) \geq d[y] + W(y, z)$

Since distance from z_k to z is non-negative, therefore, **W(P) <= W(P').**



- Design of a counter-example:
  - Look at different edges across the border
  - Design a case that violates the proof

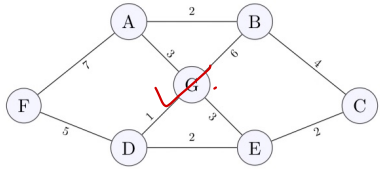


What we have exercised

- Single-source shortest path algorithm: Dijkstra's algorithm
  - Running of the algorithm
  - Data structure contents in each iteration
  - Modify it to capture more information
  - Design counter-example for its correctness

Week 8

Q4 Execute by hand the Prim's algorithm for finding minimum spanning tree (MST) on the graph in Figure 2.1, starting from vertex G. Show the contents of arrays S, d and pi after each iteration of the while loop when a vertex is added to the MST.



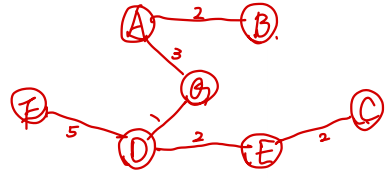


Figure 2.1: Graph for Q4

Start from G

	A	B	C	D	E	F	G
S	0	0	0	0	0	0	0
d	∞	∞	∞	∞	∞	∞	0
pi	–	–	–	–	–	–	NULL

(1st iter) choose G

	A	B	C	D	E	F	G
S	0	0	0	0	0	0	1
d	3	6	∞	1	3	∞	0
pi	G	G	NULL	G	G	NULL	NULL

↓ choose D

	A	B	C	D	E	F	G
S	0	0	0	1	0	0	1
d	3	6	∞	1	2	5	0
pi	G	G	NULL	G	D	D	G

actual weight, not added path distance ≠ Dijsktra.

choose E

	A	B	C	D	E	F	G
S	0	0	0	1	1	0	1
d	3	6	2	1	2	5	0
pi	G	G	E	G	D	D	G

↓ choose C

	A	B	C	D	E	F	G
S	0	0	1	1	1	0	1
d	3	6	2	1	2	5	0
pi	G	G	E	G	D	D	G

choose A

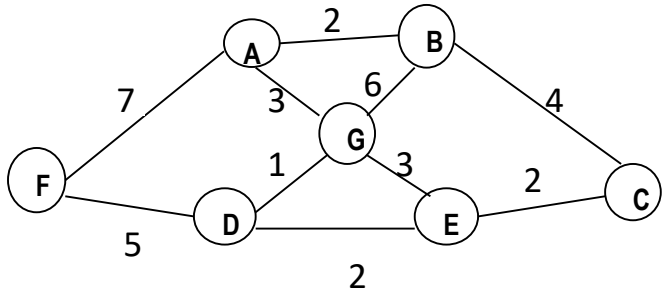
	A	B	C	D	E	F	G
S	1	0	1	1	1	0	1
d	3	2	2	1	2	5	0
pi	G	A	E	G	D	D	G

↓ choose B

	A	B	C	D	E	F	G
S	1	1	1	1	1	0	1
d	3	2	2	1	2	5	0
pi	G	A	E	G	D	D	G

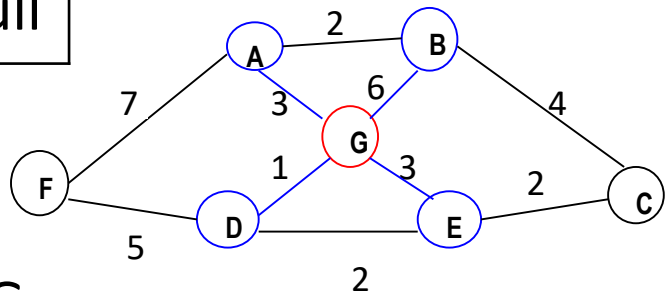
choose F

	A	B	C	D	E	F	G
S	1	1	1	1	1	0	1
d	3	2	2	1	2	5	0
pi	G	A	E	G	D	D	G



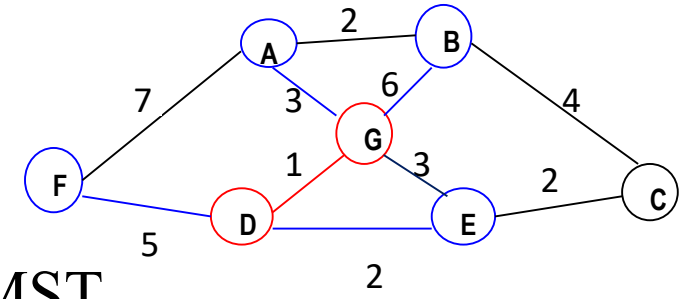
After initialization:

	A	B	C	D	E	F	G
S	0	0	0	0	0	0	1
d	∞	∞	∞	∞	∞	∞	**0**
pi	null	Null	Null	Null	Null	Null	Null



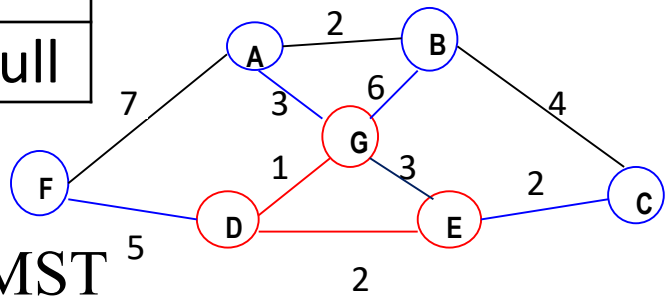
First iteration:

	A	B	C	D	E	F	G
S	0	0	0	0	0	0	1
d	**3**	**6**	∞	**1**	**3**	∞	**0**
pi	G	G	Null	G	G	Null	Null



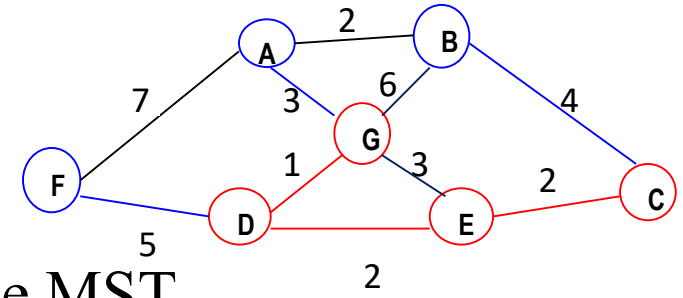
2nd iteration: Edge GD becomes part of the MST

	A	B	C	D	E	F	G
S	0	0	0	1	0	0	1
d	**3**	**6**	**∞**	**1**	**2**	**5**	**0**
pi	G	G	Null	G	D	D	Null



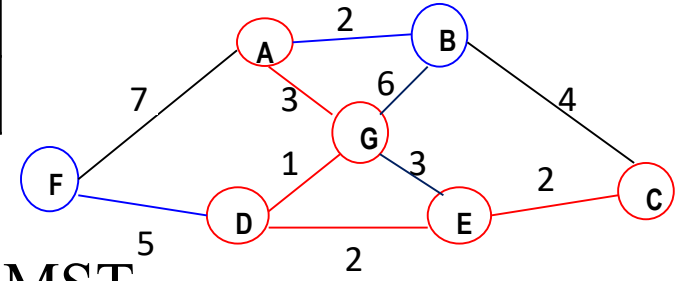
3rd iteration: Edge DE becomes part of the MST

	A	B	C	D	E	F	G
S	0	0	0	1	1	0	1
d	**3**	**6**	**2**	**1**	**2**	**5**	**0**
pi	G	G	E	G	D	D	Null



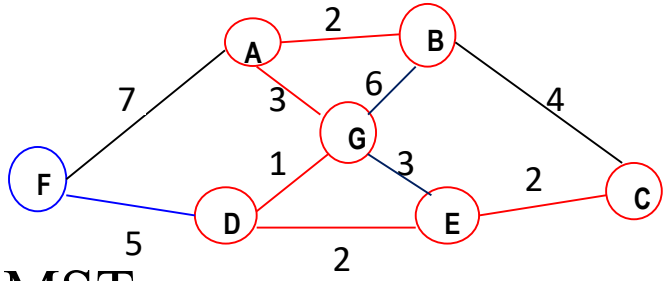
4th iteration: Edge EC becomes part of the MST

	A	B	C	D	E	F	G
S	0	0	1	1	1	0	1
d	**3**	**4**	**2**	**1**	**2**	**5**	**0**
pi	G	C	E	G	D	D	Null



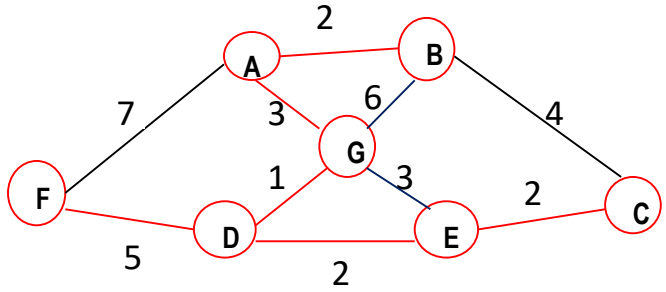
5th iteration: Edge GA becomes part of the MST

	A	B	C	D	E	F	G
S	1	0	1	1	1	0	1
d	**3**	**2**	**2**	**1**	**2**	**5**	**0**
pi	G	A	E	G	D	D	Null



6th iteration: Edge AB becomes part of the MST

	A	B	C	D	E	F	G
S	1	1	1	1	1	0	1
d	**3**	**2**	**2**	**1**	**2**	**5**	**0**
pi	G	A	E	G	D	D	Null

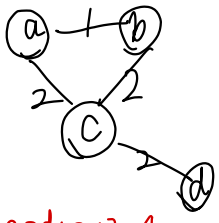


7th iteration: Edge DF becomes part of the MST

	A	B	C	D	E	F	G
S	1	1	1	1	1	1	1
d	**3**	**2**	**2**	**1**	**2**	**5**	**0**
pi	G	A	E	G	D	D	Null

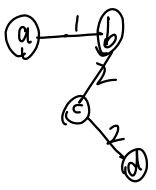
Q5 In a weighted undirected graph, is the path between two vertices in a minimum spanning tree always the shortest path (i.e. a path with the minimum weight) between the two vertices in the graph? If your answer is yes, give a proof; otherwise, give a counterexample.

No. ex1: input graph



a→d: 2+2=4.

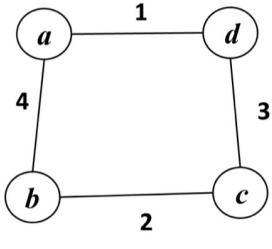
MST



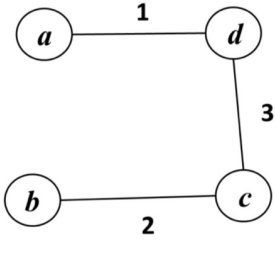
a→d: 1+2+2=5

Answer: No, it is not always the case.

A counterexample:



A weighted graph



Minimum Spanning Tree (MST)

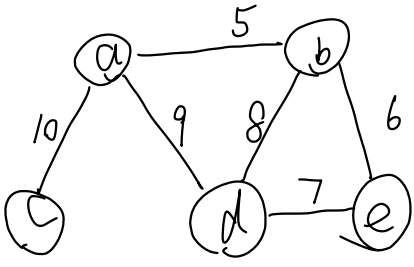
- The path from vertex *a* to vertex *b* in the MST is (*a, d, c, b*) with weight 1 + 3 + 2 = 6.
- But the shortest path from *a* to *b* in the graph is (*a, b*) with weight 4, shorter than the path in the MST.

Question 6

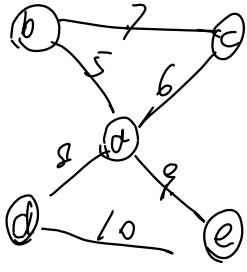
Draw a connected graph with five nodes, six edges of respective weights 5, 6, 7, 8, 9, 10, and a minimum spanning tree of weight 28. Is it possible to have an MST of weight 29? If yes, draw the graph; otherwise, provide your justification.

Q6 Draw a connected graph with five nodes, six edges of respective weights 5, 6, 7, 8, 9, 10, and a minimum spanning tree of weight 28. Is it possible to have an MST of weight 29? If yes, draw the graph; otherwise, provide your justification.

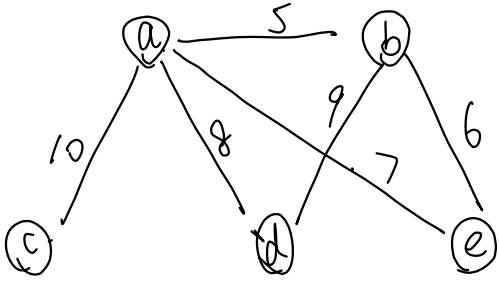
i) $28 = 5+6+7+10$



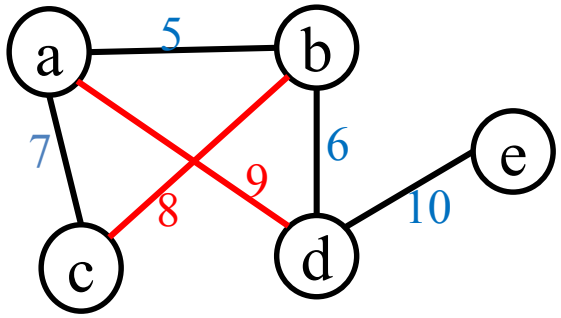
常见:

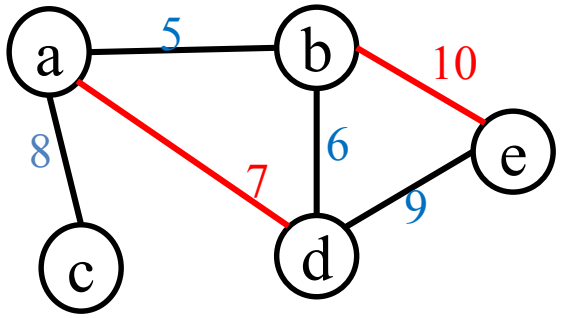


ii) $29 = 5+6+8+10$

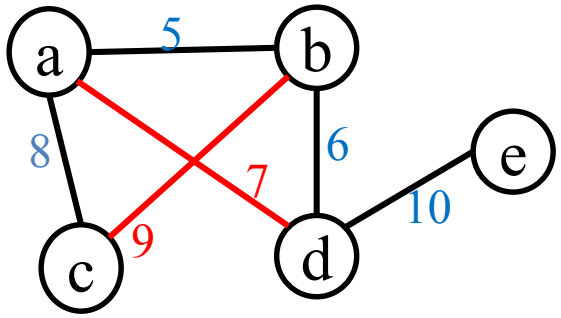


- Edge weights: 5, 6, 7, 8, 9, 10
- MST of weight 28: 5, 6, 7, 10 or 5, 6, 8, 9





- Edge weights: 5, 6, 7, 8, 9, 10
- MST of weight 29: 5, 6, 8, 10



What we have exercised

- Minimum spanning tree algorithm
  - Prim's algorithm
    - Running of the algorithm
    - Data structure contents in each iteration
    - Path in MST may not be a shortest path in G
- Minimum spanning tree – concept
  - Understand which edges may or may not be in an MST

Week 8 (Q7-Q9):

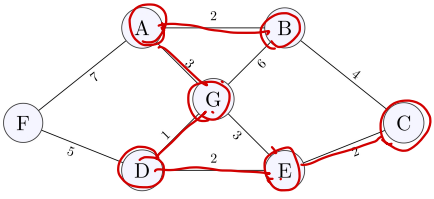
Q7 Execute by hand the Kruskal's algorithm (with the weighted QuickUnion algorithm for Union-Find) for finding minimum spanning tree (MST) on the graph in Figure 2.1. Show the contents of arrays id and sz at each step when an edge is added to the MST.

Q8 If the input graph to the Kruskal's algorithm is given in an adjacency matrix, what is the time complexity of the algorithm?

Q9 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 $|E|$.

Wek 8

Q7 Execute by hand the Kruskal's algorithm (with the weighted QuickUnion algorithm for Union-Find) for finding minimum spanning tree (MST) on the graph in Figure 2.1. Show the contents of arrays *id* and *sz* at each step when an edge is added to the MST.



1) Sort edge in an increasing order

D-G 1	✓	iter1	G-E 3	✗	adr in
D-E 2	✓	iter2	B-C 4	✗	adr in
E-C 2	✓	iter3	F-D 5	✓	
A-B 2	✓	iter4	G-B 6		
A-G 3	✓	iter5	A-F 7		

- Data structure:
  - Extra array sz[i] to count no. of objects in the tree rooted at i.

ID: ID of tree root.

tree root of A is A

	A	B	C	D	E	F	G
id.	A	B	C	D	E	F	G
SZ.	1	1	1	1	1	1	1

W A为root 那树有1个节点.

1st Iter

	A	B	C	D	E	F	G.
ID	A	B	C	D	E	F	D
SZ	1	1	1	2	1	1	1

2nd Iter

	A	B	C	D	E	F	G.
ID	A	B	C	D	D	F	D
SZ	1	1	1	3.	1	1	1

3rd Iter

	A	B	C	D	E	F	G.
ID	A	B	D	D	D	F	D
SZ	1	1	1	4.	1	1	1

4th Iter

	A	B	C	D	E	F	G.
ID	A	A	D	D	D	F	D
SZ	2	1	1	4.	1	1	1

5th Iter

	A	B	C	D	E	F	G.
ID	D	A	D	D	D	F	D
SZ	2	1	1	6.	1	1	1

6th Iter

	A	B	C	D	E	F	G.
ID	D	A	D	D	D	F	D
SZ	2	1	1	6.	1	1	1

7th Iter

	A	B	C	D	E	F	G.
ID	D	A	D	D	D	F	D
SZ	2	1	1	6.	1	1	1

no update, because already in union.

8th Iter

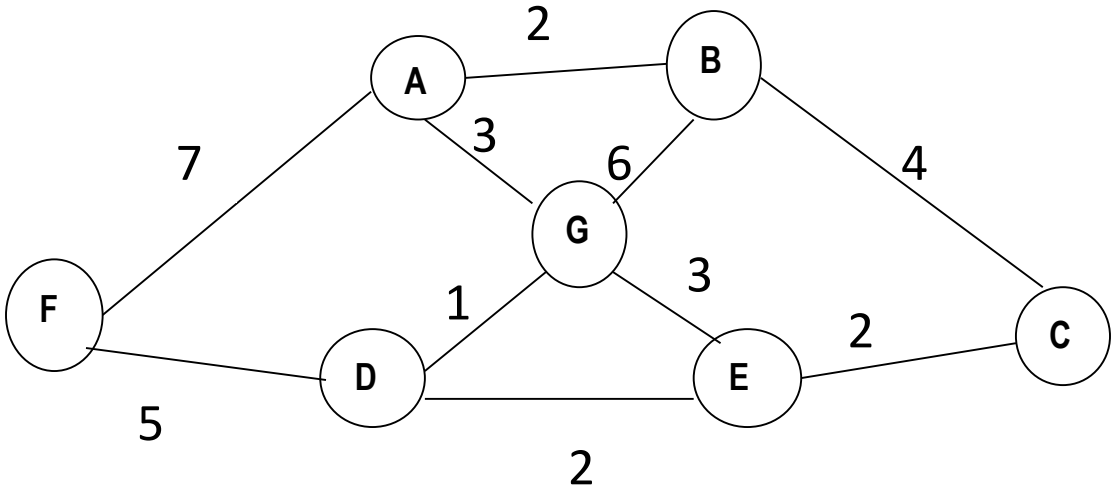
	A	B	C	D	E	F	G.
ID	D	A	D	D	D	D	D
SZ	2	1	1	7	1	1	1

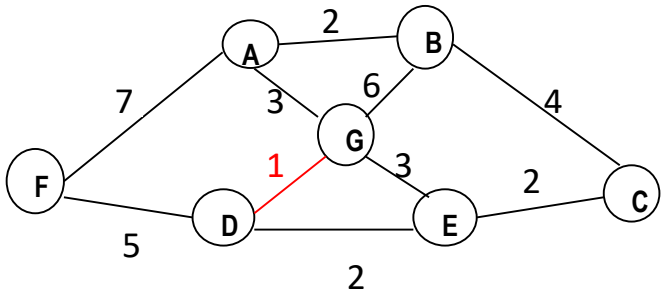
```
public class KruskalMST
{
  private Queue<Edge> mst = new Queue<Edge>();

  public KruskalMST(EdgeWeightedGraph G)
  {
    MinPQ<Edge> pq = new MinPQ<Edge>(G.edges());   O(|E|)
    UF uf = new UF(G.V());                         O(|V|)
    while (!pq.isEmpty() && mst.size() < G.V()-1)
    {
      Edge e = pq.delMin();                        O(|E| log|E|)
      int v = e.either(), w = e.other(v);
      if (!uf.connected(v, w))                     O(|E| log*|V|)
      {
        uf.union(v, w);                            O(|V| log*|V|)
        mst.enqueue(e);                            O(|V|)
      }
    }
  }                                                Overall: O(|E| log|E|)
  public Iterable<Edge> edges()
  { return mst; }
}
```

Question 7

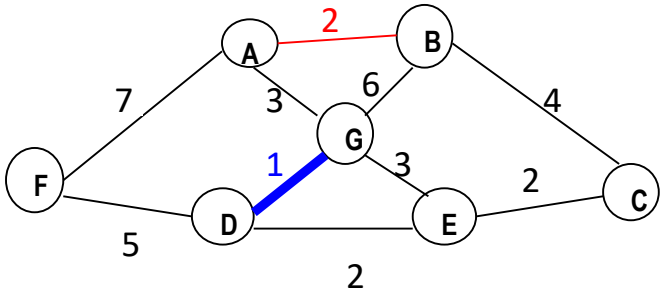
- Execute by hand the Kruskal's algorithm (with the weighted QuickUnion algorithm for Union-Find) for finding minimum spanning tree (MST) on the graph below. Show the contents of arrays *id* and *sz* at each step when an edge is added to the MST.





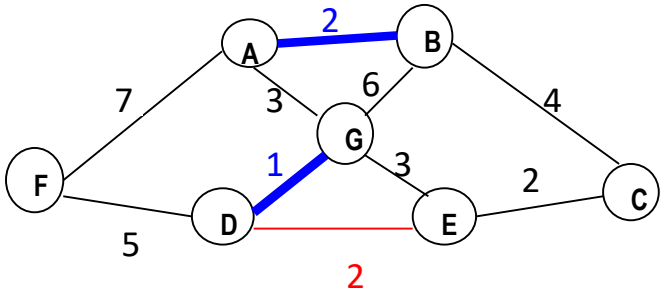
After initialization:

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



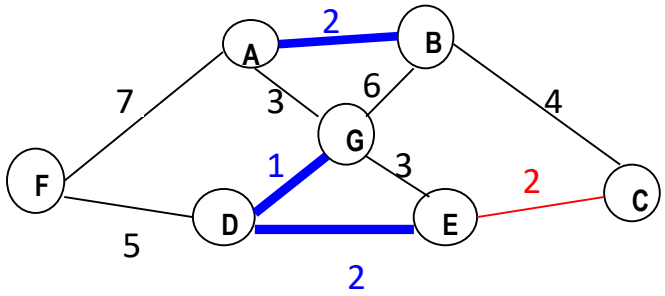
1st iteration:

	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:

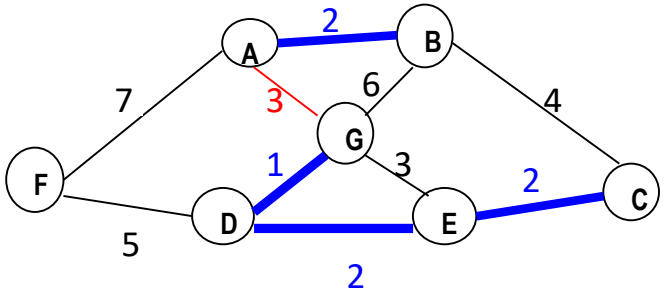
	A	B	C	D	E	F	G
id	A	A	C	D	E	F	D
sz	**2**	**1**	**1**	**2**	**1**	**1**	**1**



3rd iteration:

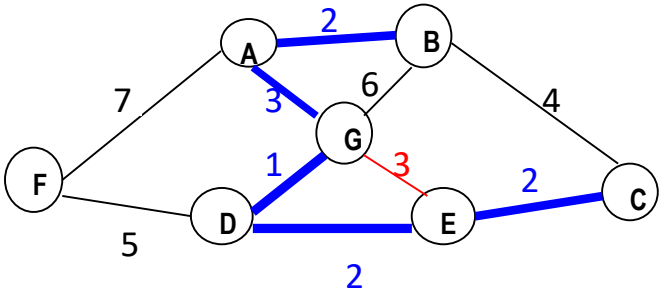
	A	B	C	D	E	F	G
id	A	A	C	D	D	F	D
sz	**2**	**1**	**1**	**3**	**1**	**1**	**1**

4th iteration:



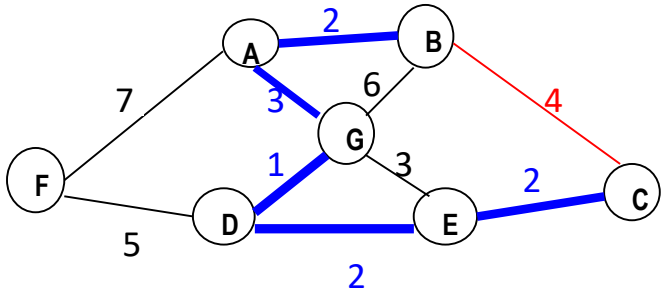
	A	B	C	D	E	F	G
id	A	A	D	D	D	F	D
sz	**2**	**1**	**1**	**4**	**1**	**1**	**1**

5th iteration:



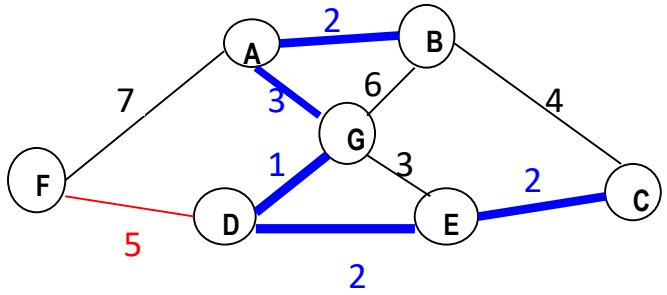
	A	B	C	D	E	F	G
id	D	A	D	D	D	F	D
sz	**2**	**1**	**1**	**6**	**1**	**1**	**1**

6th iteration:



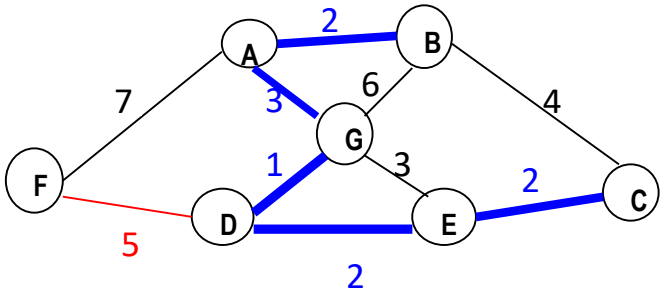
	A	B	C	D	E	F	G
id	D	A	D	D	D	F	D
sz	**2**	**1**	**1**	**6**	**1**	**1**	**1**

7th iteration:

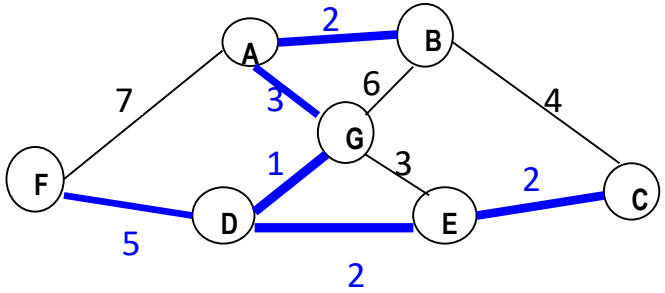


	A	B	C	D	E	F	G
id	D	A	D	D	D	F	D
sz	**2**	**1**	**1**	**6**	**1**	**1**	**1**

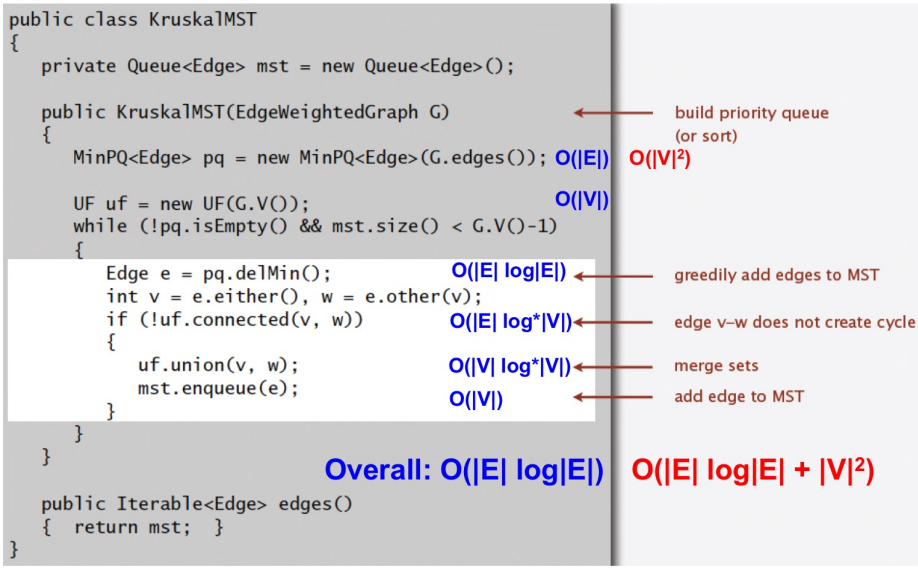
8th iteration:



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



Q8 If the input graph to the Kruskal's algorithm is given in an adjacency matrix, what is the time complexity of the algorithm?



Q9 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|)