Data Structures and Algorithms
(ES221)

Graph Algorithms

Dr. Zubair Ahmad

Depth First Search (DFS)

- Formally, DFS is an uninformed search that progresses by
  - expanding the first child node of the search tree (or graph) that appears, and
  - thus going deeper and deeper
  - until a goal node is found, or until it hits a node that has no children.
- Then the search backtracks, returning to the most recent node it hasn't finished exploring.
- In a non-recursive implementation, all freshly expanded nodes are added to a stack for exploration.

DFS Pseudocode

procedure DFS(*G*, *s*)

    for each vertex *u* in *G*

        u.status = **notVisited/white**

        u.pi = *NULL*

        time = 0;

    for each vertex *u* in *G*

        if *u.status* = **notVisited/white**

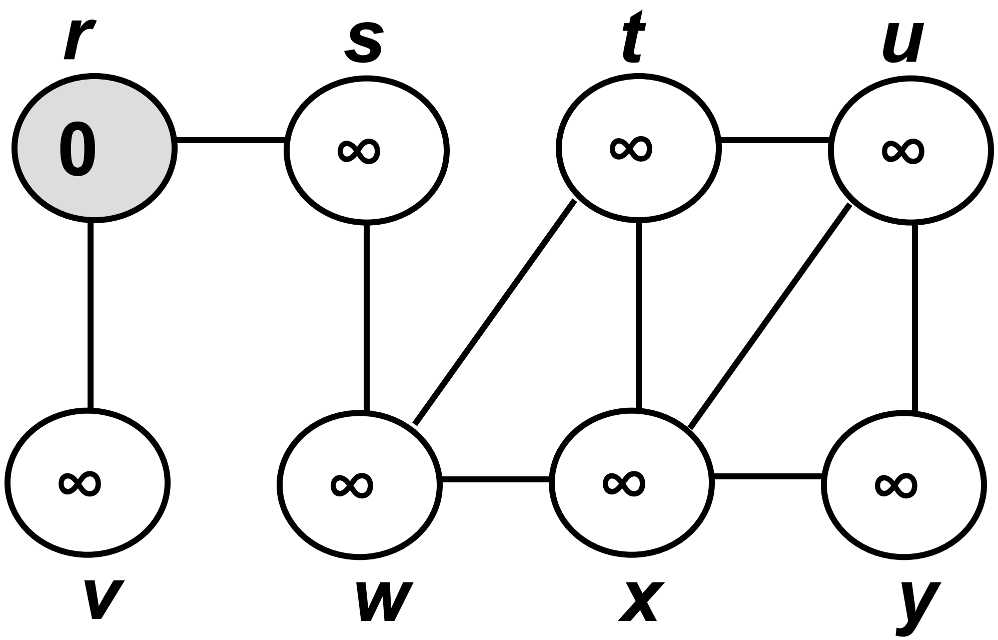
                DFS_visit(*G*, *u*)

DFS Pseudocode

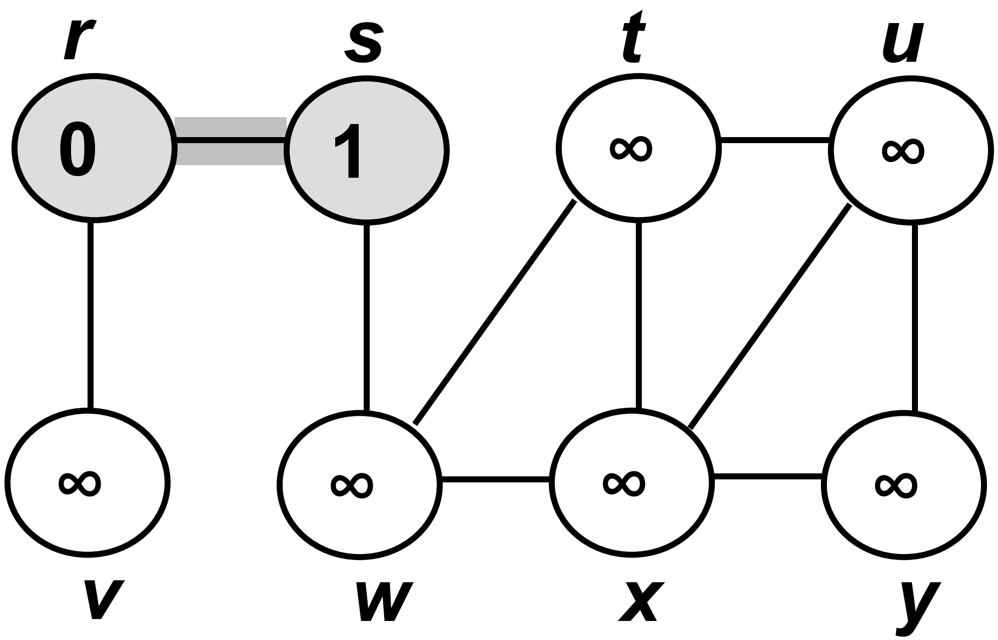
```
procedure DFS_visit(G, u)
    u.status = visited / grey
      time = time + 1;
      u.d = time;
    for each v adjacent to u do
        if v.status = notVisited/white
            v.pi = u;
            DFS_visit(G, v)
    u.status = processed/ black
```


4

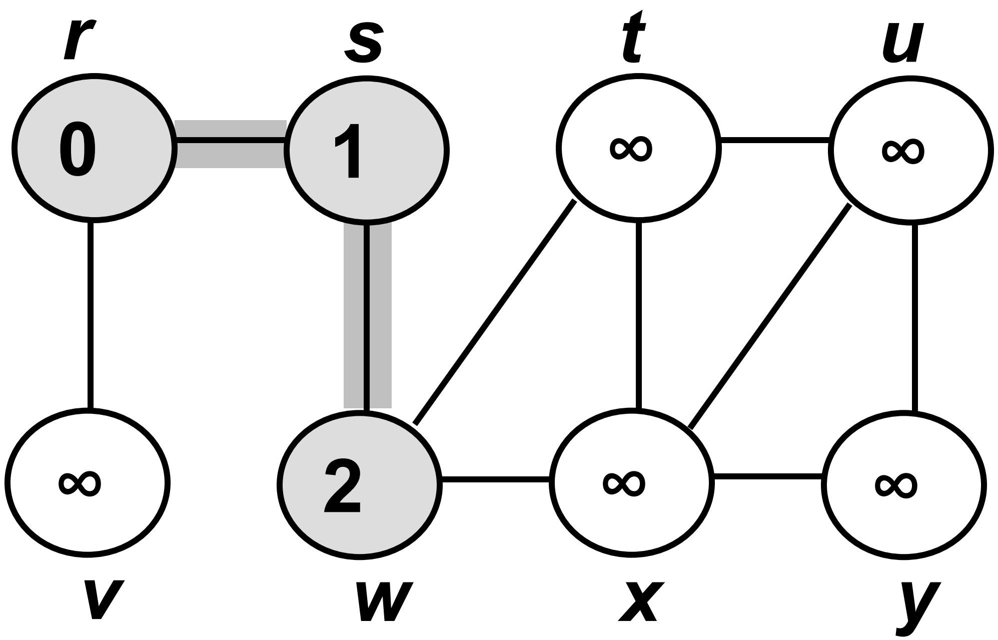
DFS Example



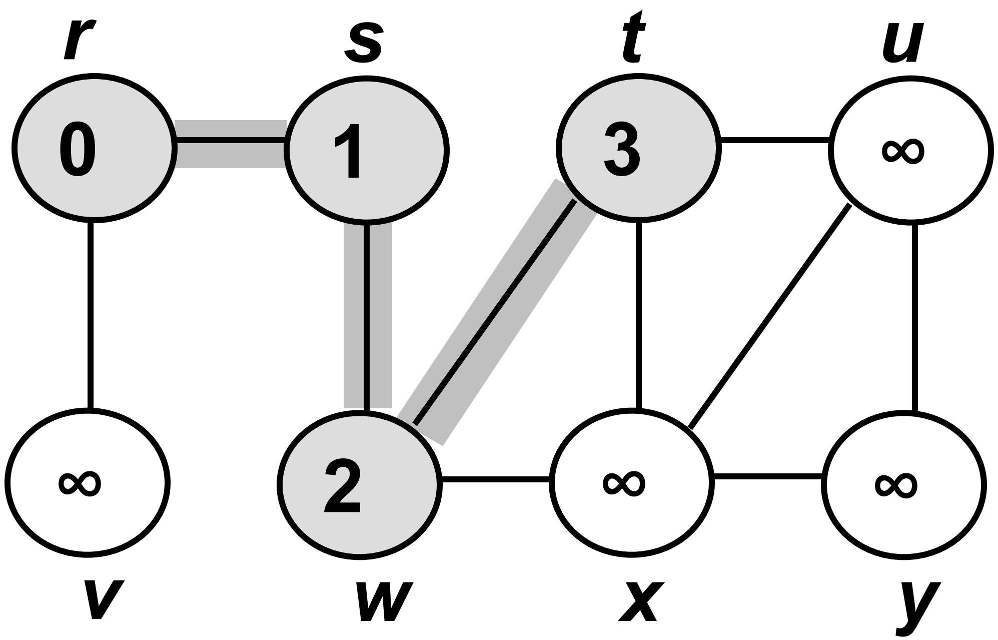
DFS Example



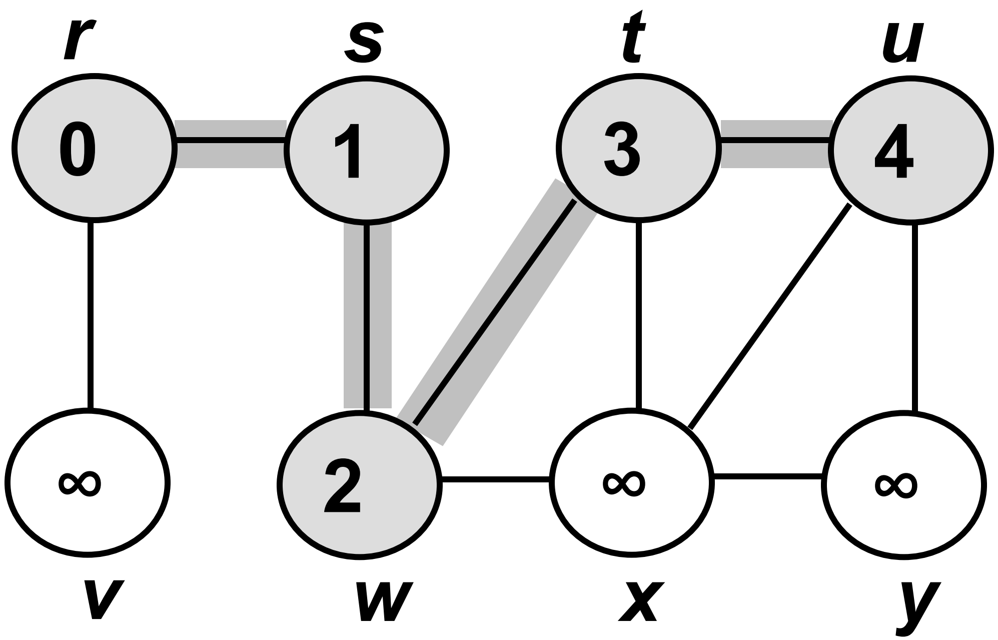
DFS Example



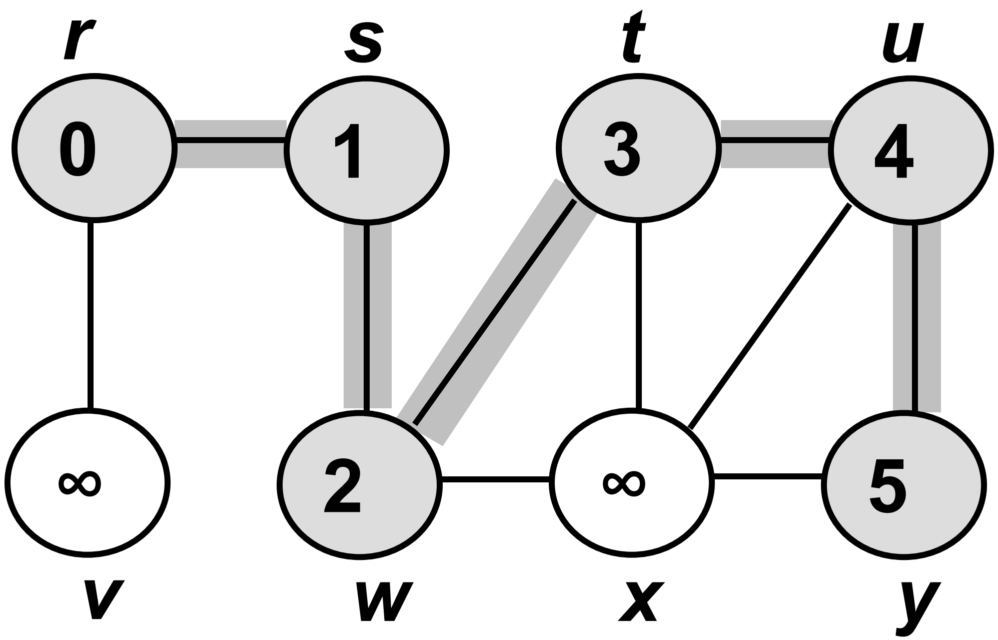
DFS Example



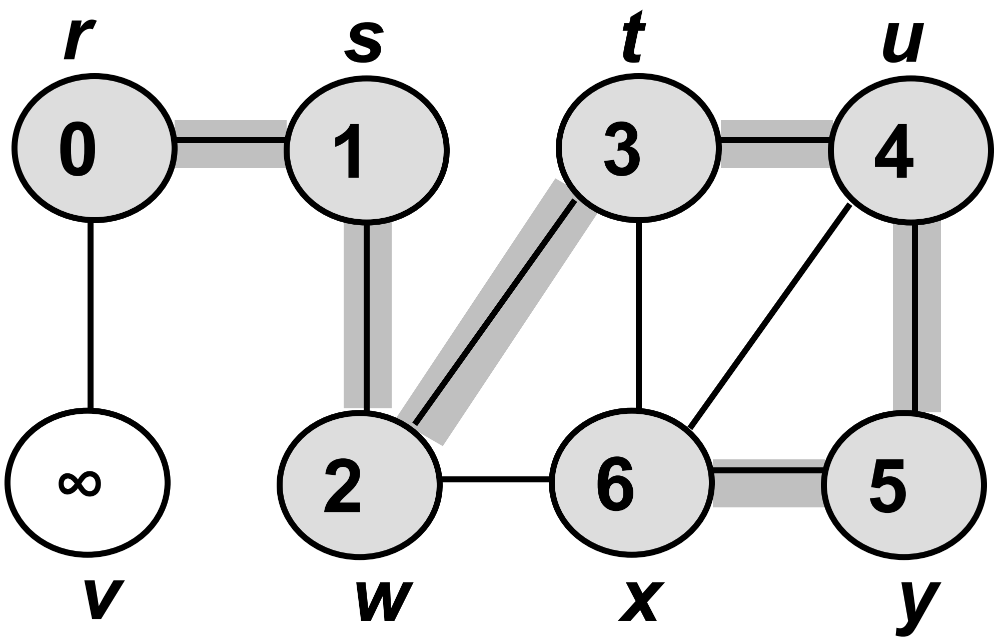
DFS Example



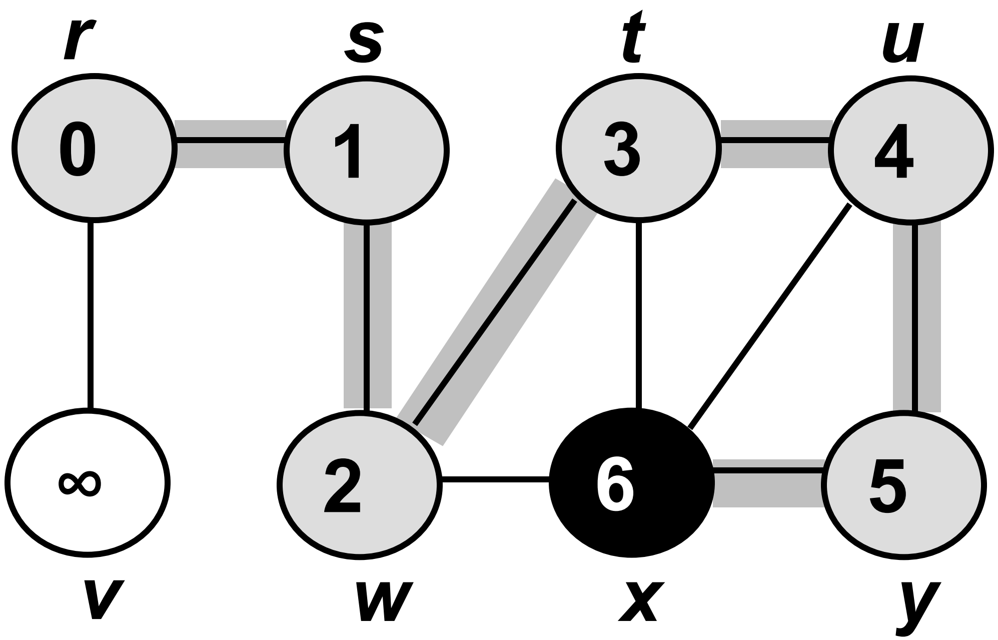
DFS Example



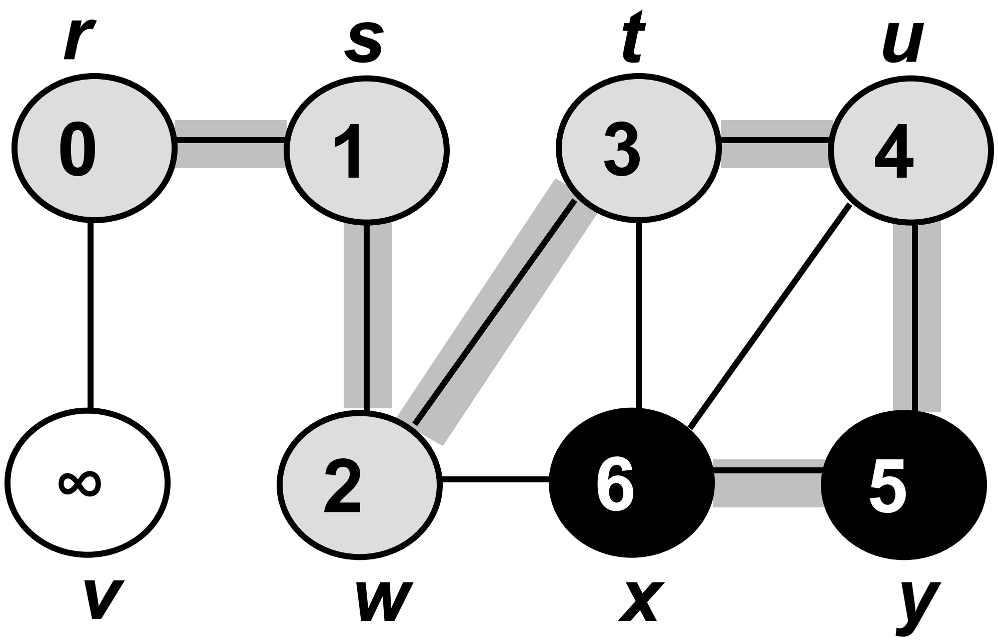
DFS Example



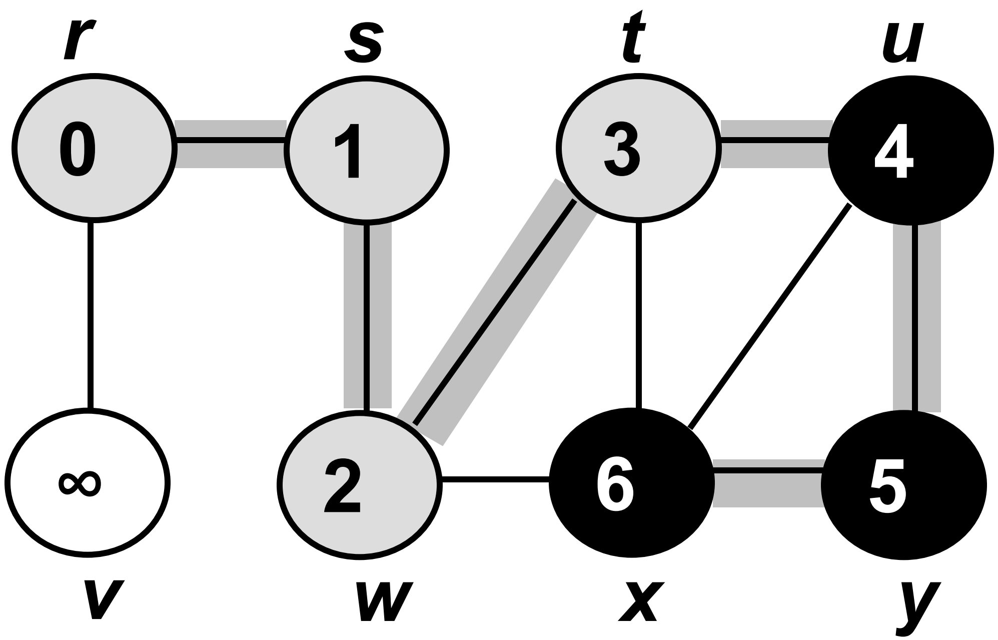
DFS Example



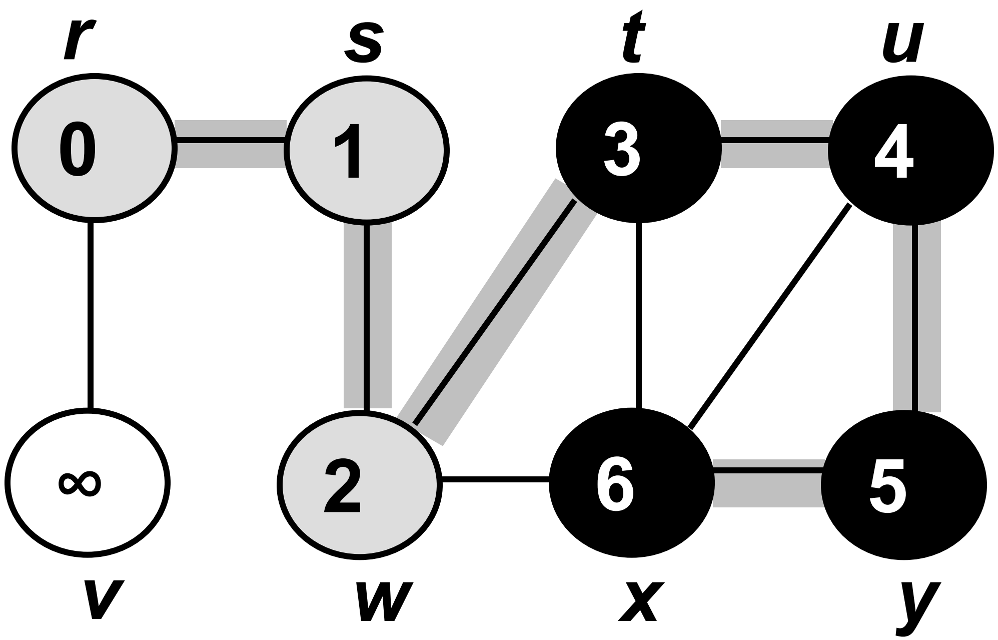
DFS Example



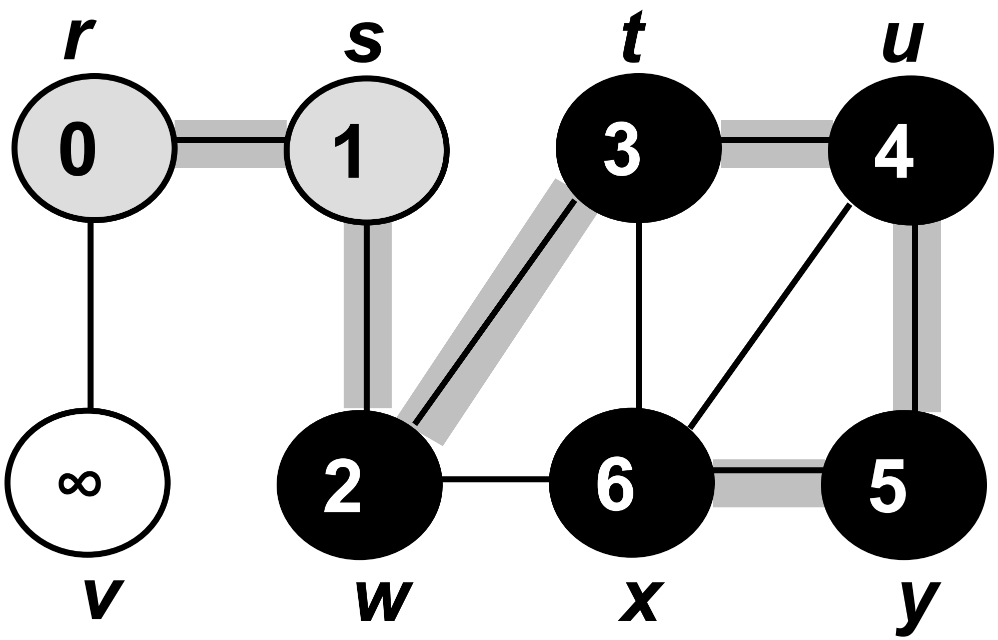
DFS Example



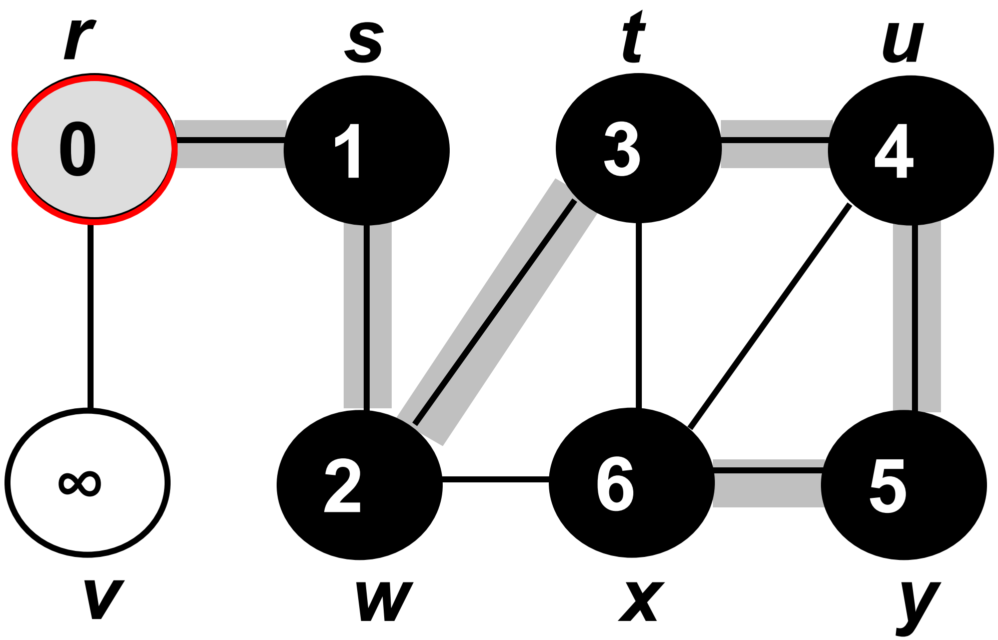
DFS Example



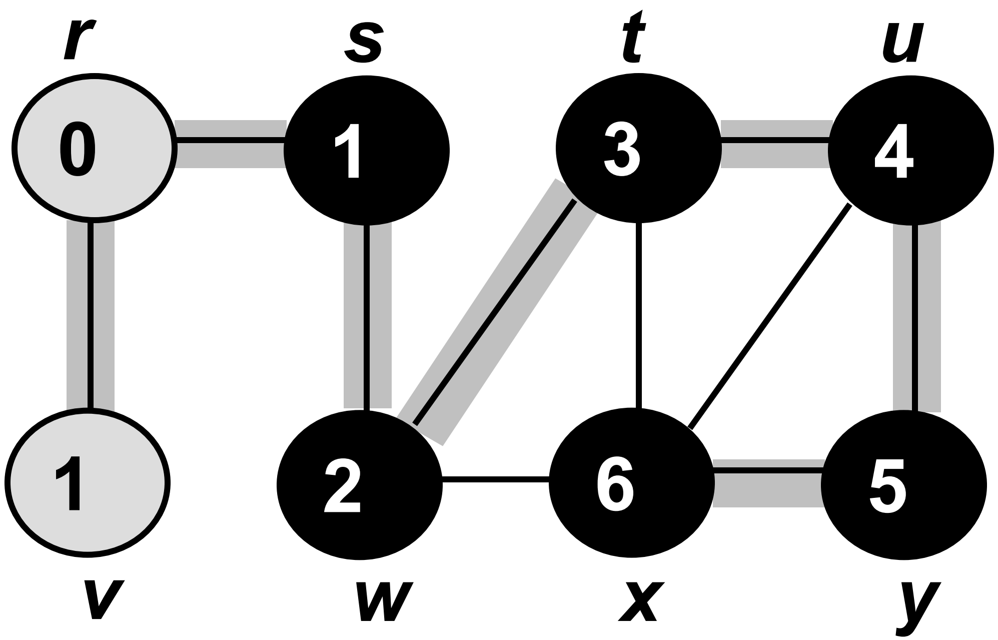
DFS Example



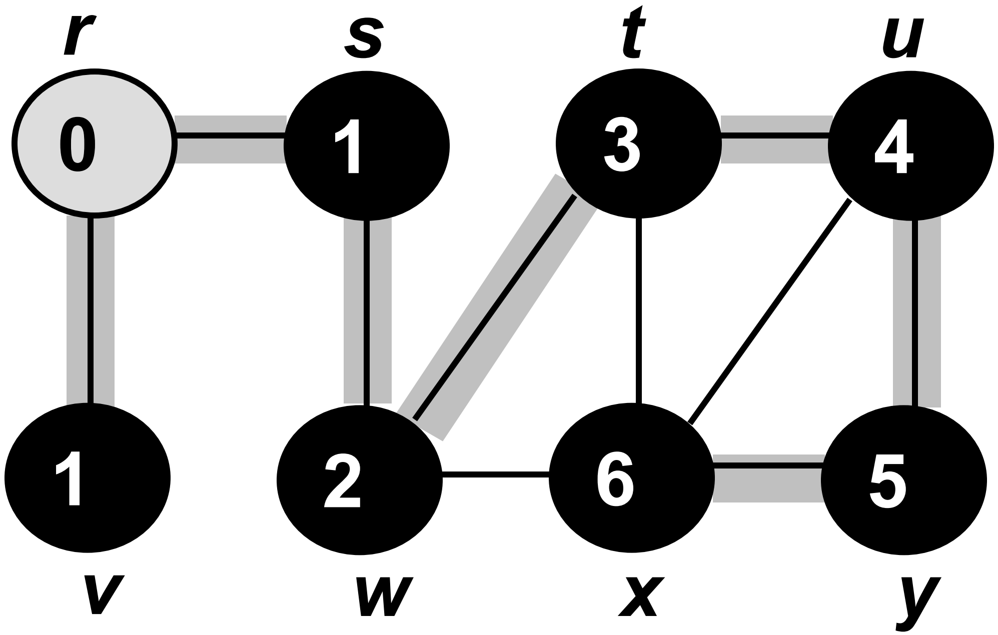
DFS Example



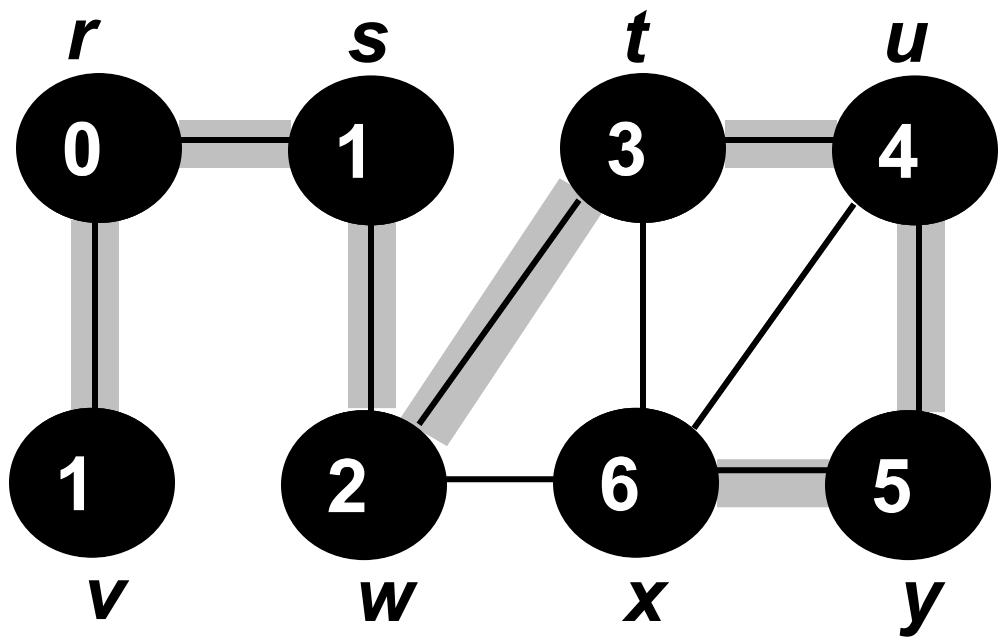
DFS Example



DFS Example



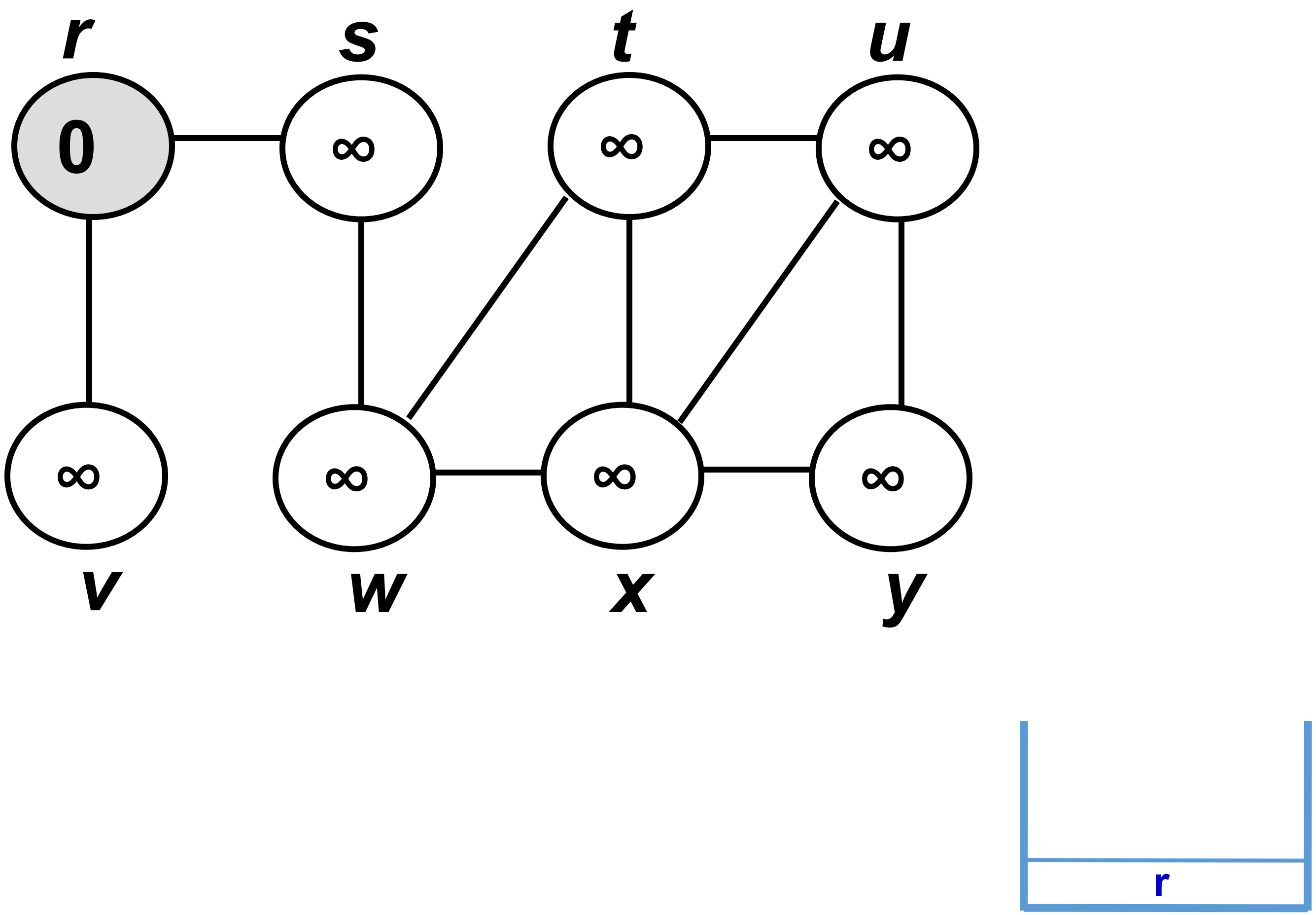
DFS Example



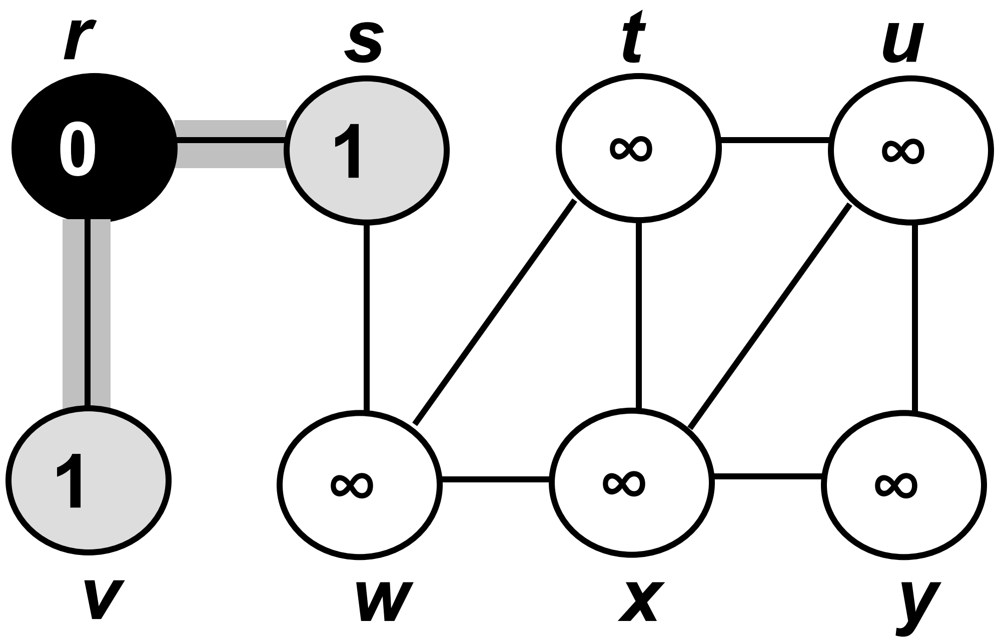
DFS: Non-recursive Implementation

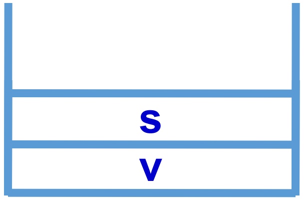
1. Initialize all nodes to the notVisited state (STATUS=1)
2. PUSH starting node A on stack and change its status to visited(STATUS=2)
3. Repeat 4-5 until stack is empty
    4. POP *N*. Process it and change its status to processed(STATUS=3)
    5. PUSH on stack all the neighbors of *N* that are still in notVisited state and change their status to visited
6. Exit

DFS Example

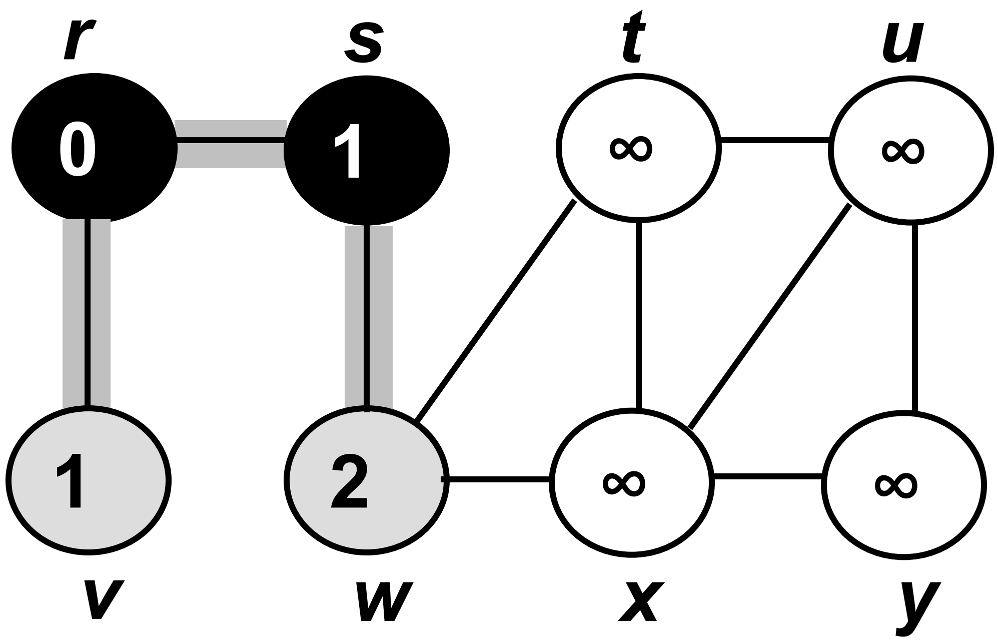


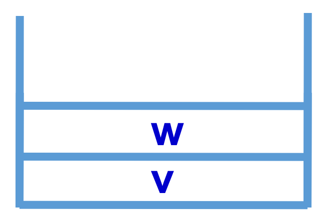
DFS Example



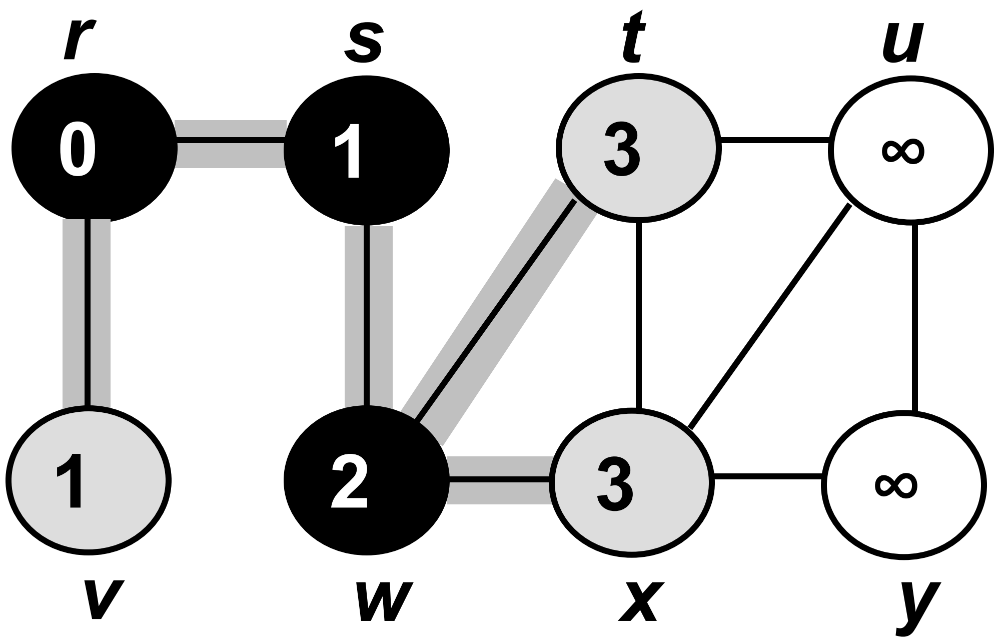


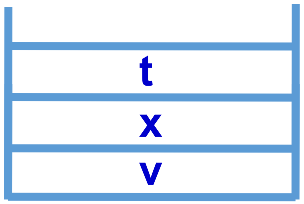
DFS Example



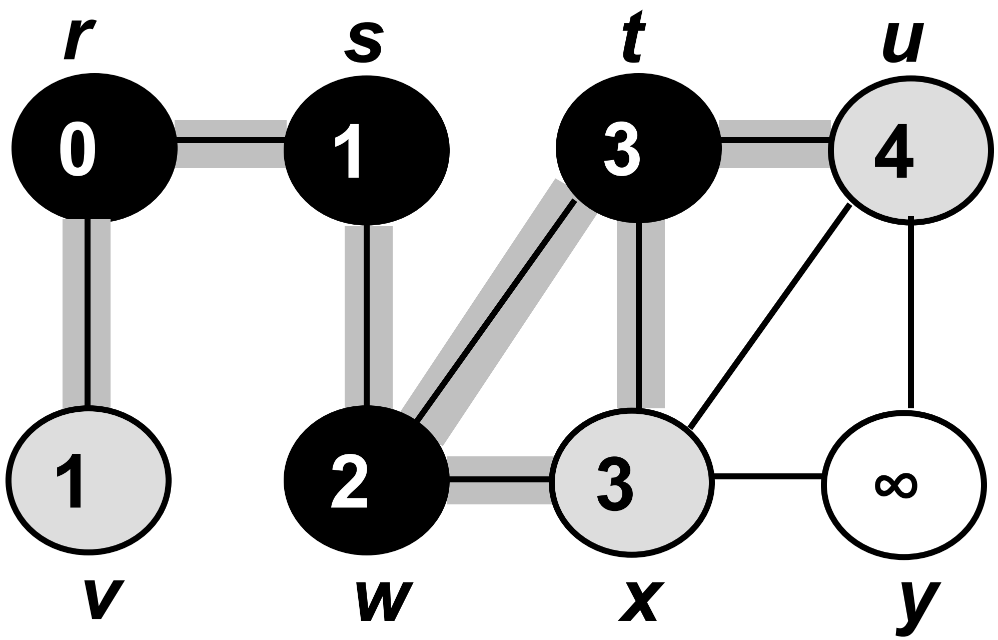


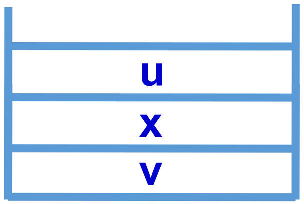
DFS Example



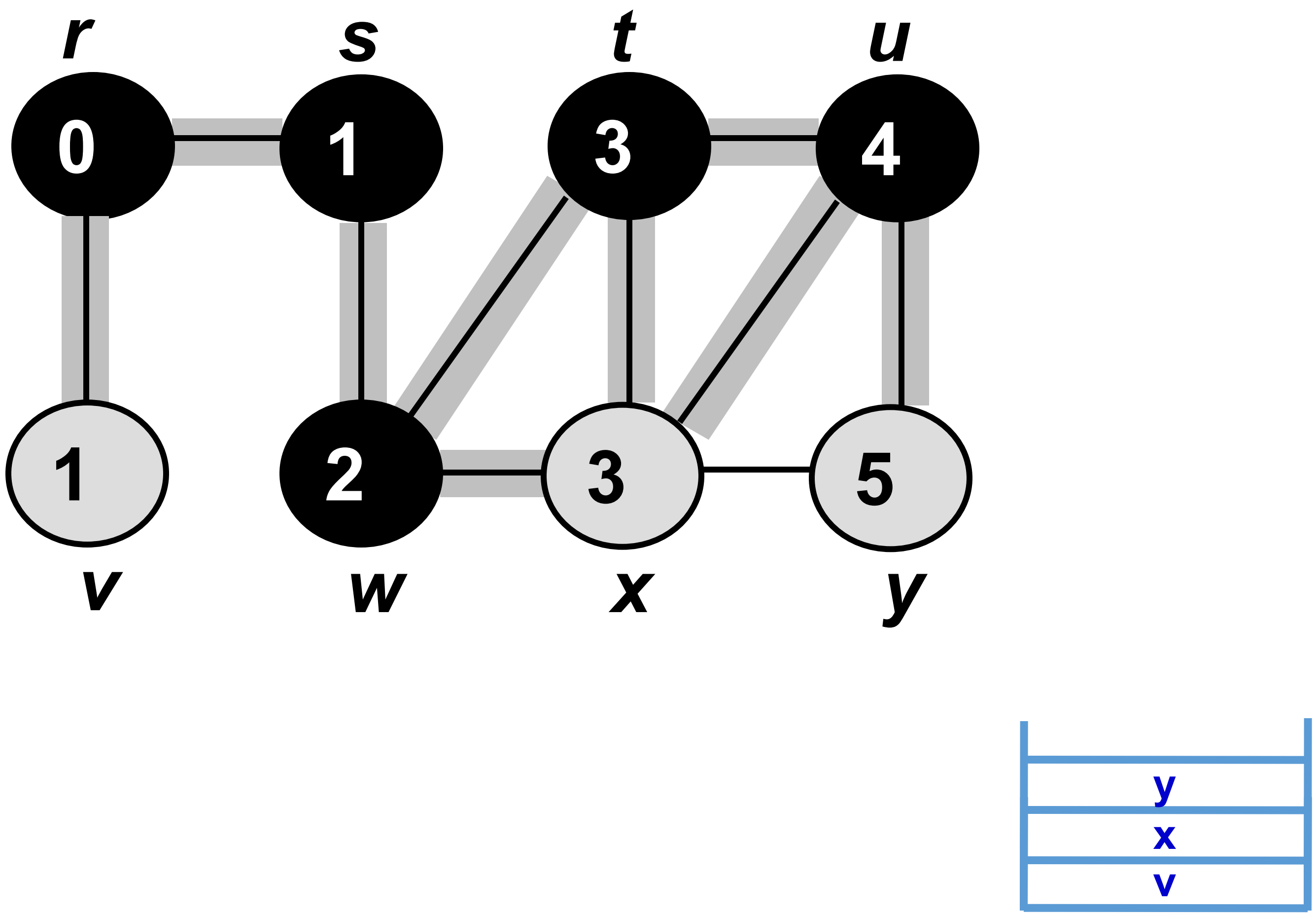


DFS Example

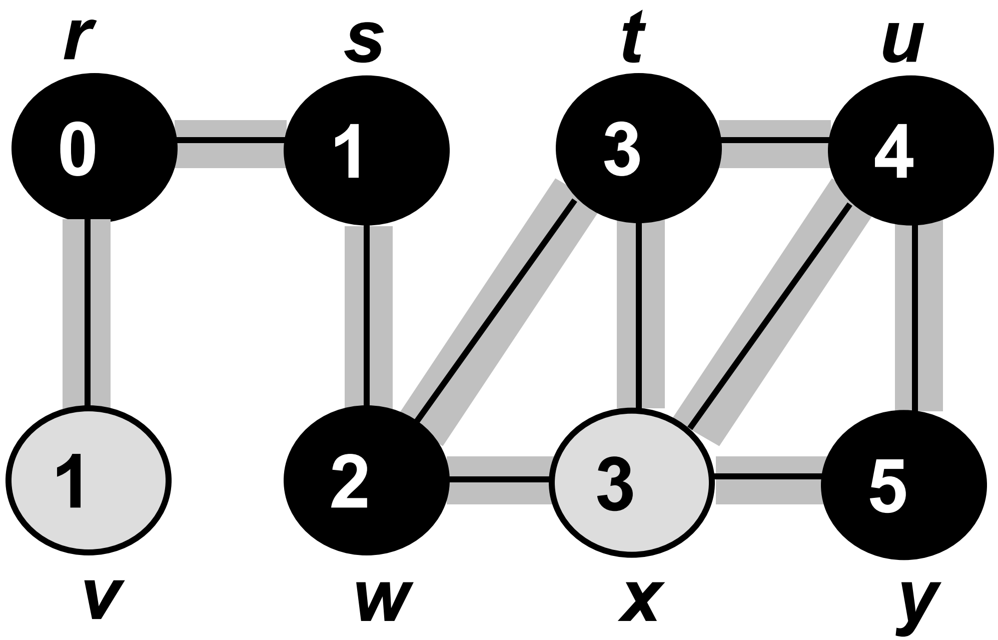


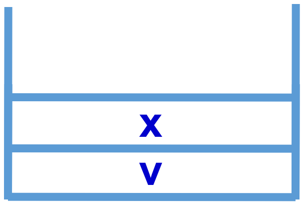


DFS Example

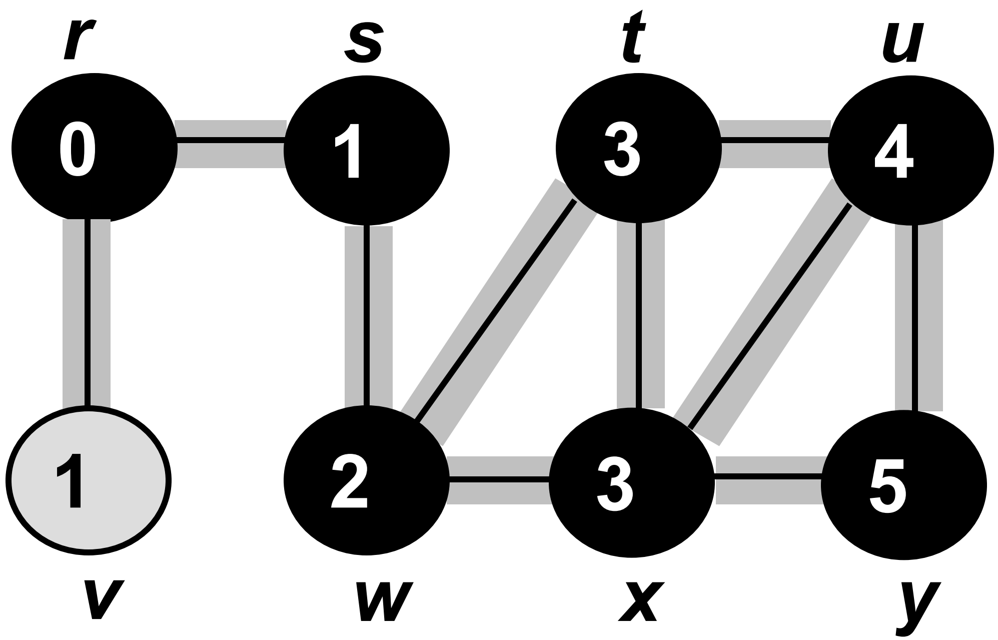


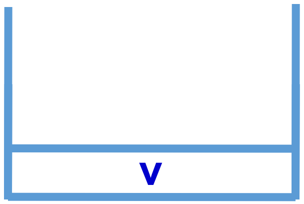
DFS Example



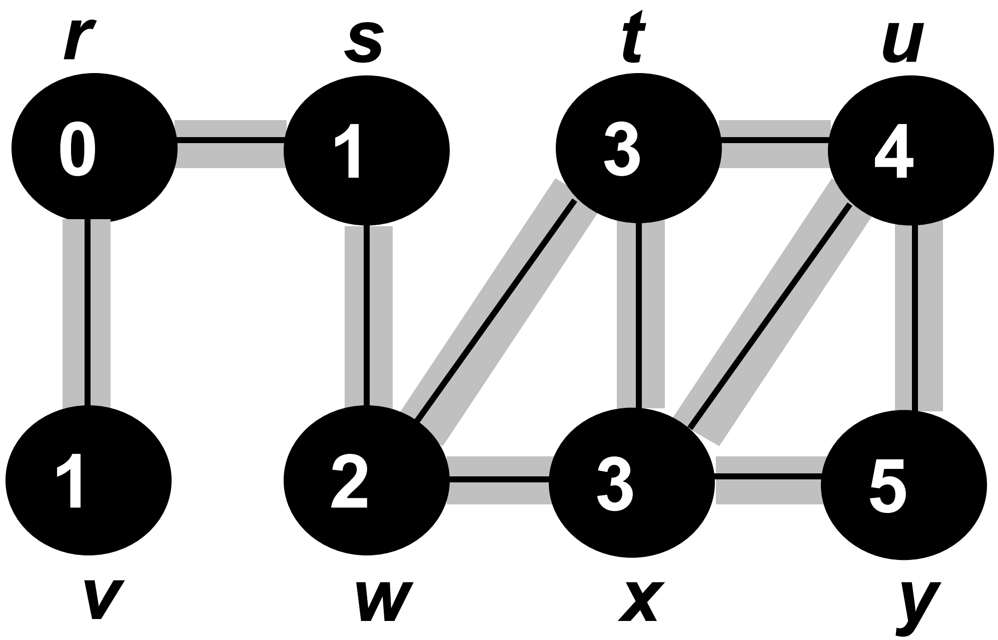


DFS Example





DFS Example



Dijkstra’ s algorithm

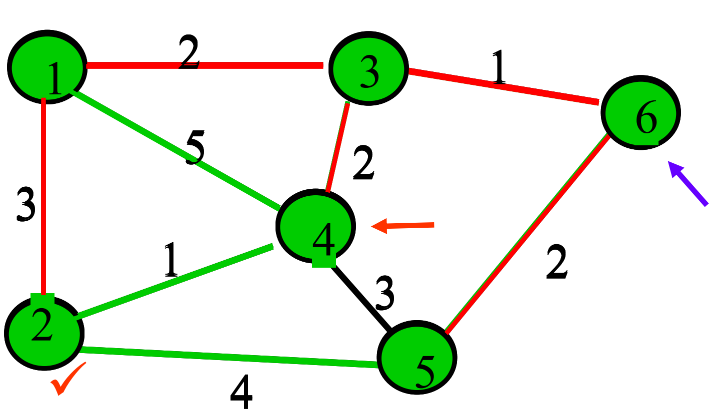
The **Dijkstra algorithm** is a **greedy algorithm** used to find the **shortest paths from a single source node** to all other nodes in a graph with **non-negative edge weights**

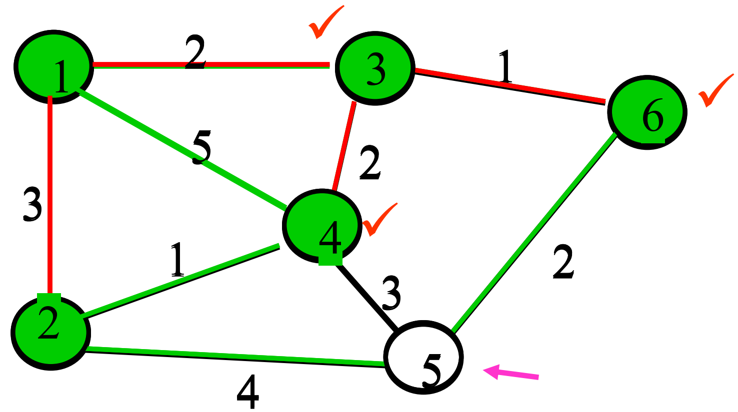
Dijkstra's algorithm

- N: set of nodes for which shortest path already found
- Initialization: (S*tart with source node s)*
  - $N = \{s\}$, $D_s = 0$, "s is distance zero from itself"
  - $D_j = C_{sj}$ for all $j \neq s$, distances of directly-connected neighbors
- Step A: (*Find next closest node i*)
  - Find $i \notin N$ such that
  - $D_i = \min Dj$ for $j \notin N$
  - Add i to N
  - If N contains all the nodes, stop
- Step B: (*update minimum costs)*
  - For each node $j \notin N$
  - $D_j = \min (D_j, D_i + C_{ij})$
  - Go to Step A

Minimum distance from s to j through node i in N

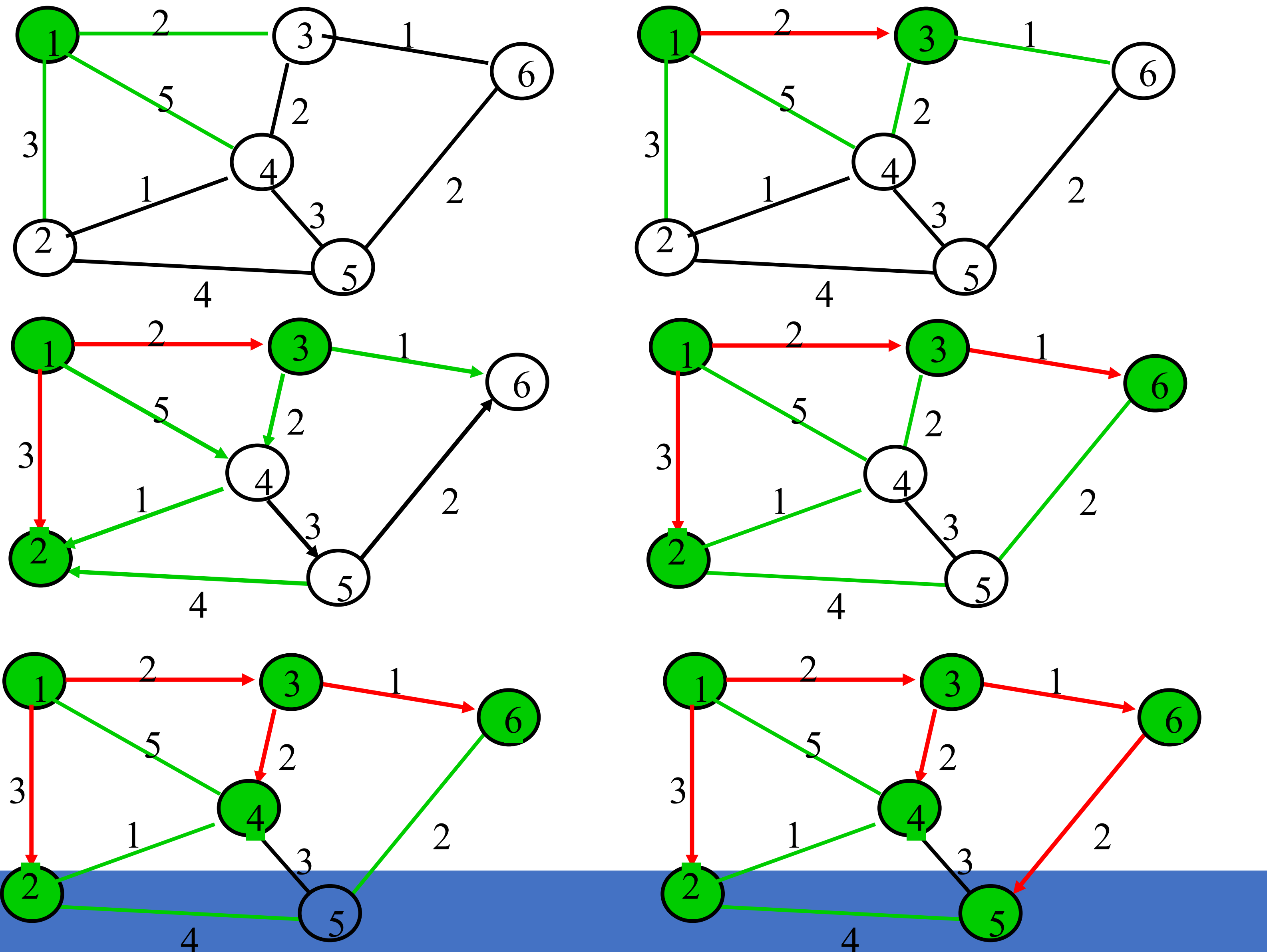
Execution of Dijkstra algorithm





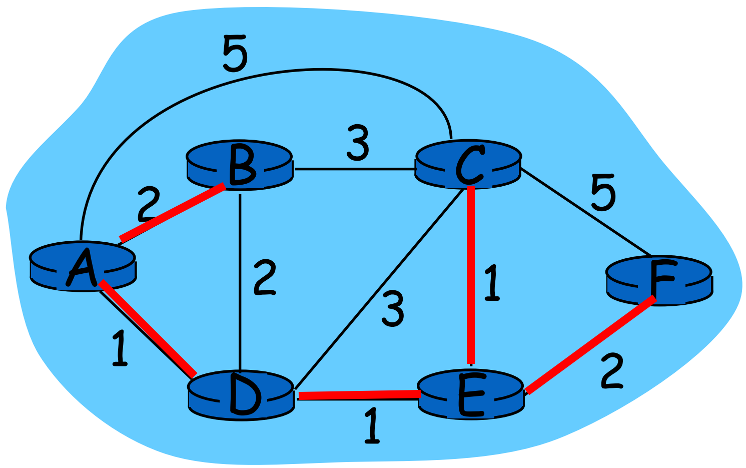
Iteration	N	D_2	D_3	D_4	D_5	D_6
Initial	{1}	3	2	5	$\propto$	$\propto$
1	{1,3}	3	**2**	4	$\propto$	3
2	{1,2,3}	**3**	**2**	4	7	3
3	{1,2,3,6}	**3**	**2**	4	5	**3**
4	{1,2,3,4,6}	**3**	**2**	**4**	5	**3**
5	{1,2,3,4,5,6}	**3**	**2**	**4**	**5**	**3**

Shortest Paths in Dijkstra's Algorithm



Dijkstra's algorithm: example

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infinity	infinity
1	AD	2,A	4,D		2,D	infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					



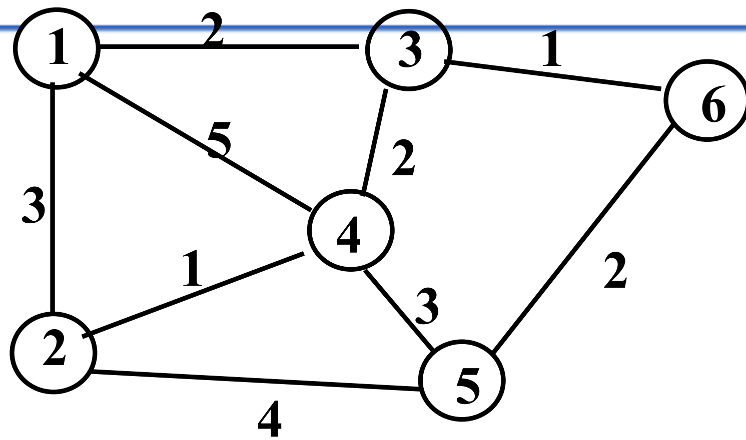
All pairs shortest path

- So far, we have discussed the problem of finding the shortest paths, starting from a specific node .
- How about finding the shorted path between every possible pair of nodes in a graph?
- Which algorithm can be used?
- Dijkstra’ s algorithm can be used
  - Call the Dijkstra’ s algorithm by setting each node as the source node, one by one
- Any Other solution?
  - Floyd -Warshal Algorithm can be used instead.

Floyd-warshall algorithm

- Initially $d_{ij} = w_{ij}$
  - $w_{ij} = 0$ if $i = j$
  - $w_{ij} =$ weight of the directed edge if $i \neq j$, and $(i, j) \in E$
  - $w_{ij} = \infty$ if $i \neq j$, and $(i, j) \notin E$
- **Initially**
  - $p_{ij} = \text{Nil}$ if $i = j$ or $w_{ij} = \infty$
  - $p_{ij} = i$ if $i \neq j$ and $w_{ij} < \infty$

Floyd-warshall algorithm



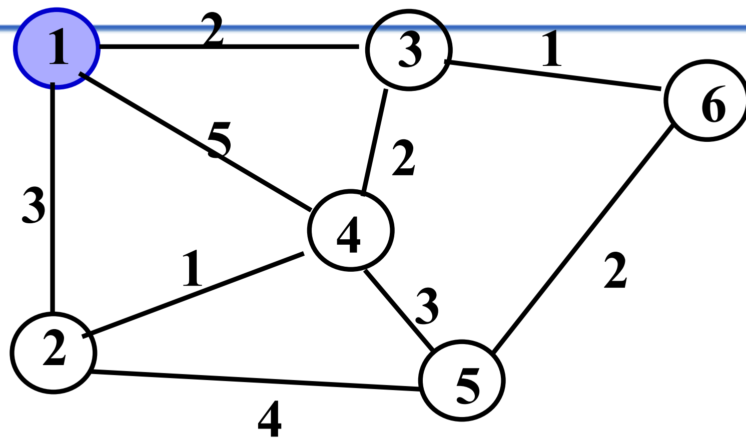
$D^{(0)}$

i / j	1	2	3	4	5	6
1	0	3	2	5	∞	∞
2	3	0	∞	1	4	∞
3	2	∞	0	2	∞	1
4	5	1	2	0	3	∞
5	∞	4	∞	3	0	2
6	∞	∞	1	∞	2	0

$P^{(0)}$

I / j	1	2	3	4	5	6
1	Nil	1	1	1	Nil	Nil
2	2	Nil	Nil	2	2	Nil
3	3	Nil	Nil	3	Nil	3
4	4	4	4	Nil	4	4
5	Nl	5	Nl	5	Nil	5
6	Nil	Nil	6	Nil	6	Nil

Floyd-warshall algorithm



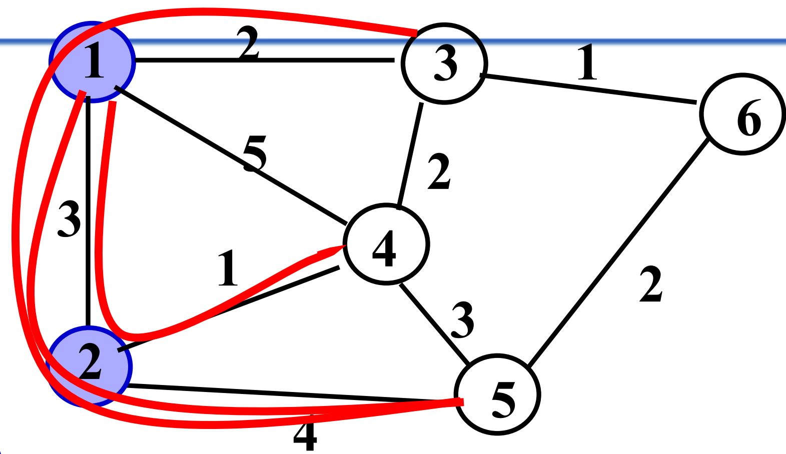
$D^{(1)}$

i / j	1	2	3	4	5	6
1	0	3	2	5	∞	∞
2	3	0	5	1	4	∞
3	2	5	0	2	∞	1
4	5	1	2	0	3	∞
5	∞	4	∞	3	0	2
6	∞	∞	1	∞	2	0

$P^{(1)}$

I / j	1	2	3	4	5	6
1	Nil	1	1	1	Nil	Nil
2	2	Nil	1	2	2	Nil
3	3	1	Nil	3	Nil	3
4	4	4	4	Nil	4	4
5	NI	5	NI	5	Nil	5
6	Nil	Nil	6	Nil	6	Nil

Floyd-warshall algorithm



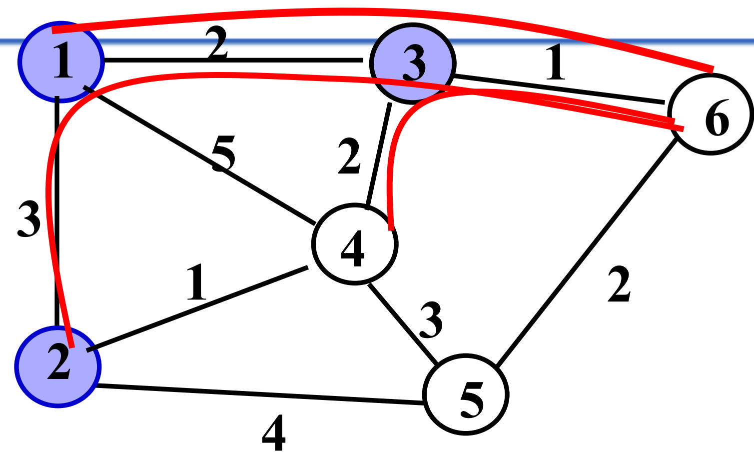
$D^{(2)}$

i / j	1	2	3	4	5	6
1	0	3	2	4	7	∞
2	3	0	5	1	4	∞
3	2	5	0	2	9	1
4	4	1	2	0	3	∞
5	7	4	9	3	0	2
6	∞	∞	1	∞	2	0

$P^{(2)}$

I / j	1	2	3	4	5	6
1	Nil	1	1	2	2	Nil
2	2	Nil	1	2	2	Nil
3	3	1	Nil	3	2	3
4	2	4	4	Nil	4	4
5	2	5	1	5	Nil	5
6	Nil	Nil	6	Nil	6	Nil

Floyd-warshall algorithm



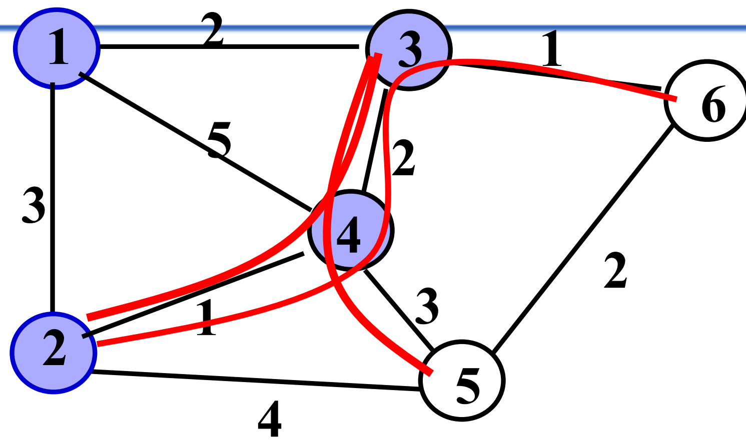
$D^{(3)}$

i / j	1	2	3	4	5	6
1	0	3	2	4	7	3
2	3	0	5	1	4	6
3	2	5	0	2	9	1
4	4	1	2	0	3	3
5	7	4	9	3	0	2
6	3	6	1	3	2	0

$P^{(3)}$

i / j	1	2	3	4	5	6
1	*Nil*	1	1	2	2	3
2	2	*Nil*	1	2	2	3
3	3	1	*Nil*	3	2	3
4	2	4	4	*Nil*	4	3
5	2	5	1	5	*Nil*	5
6	3	1	6	3	6	*Nil*

Floyd-warshall algorithm



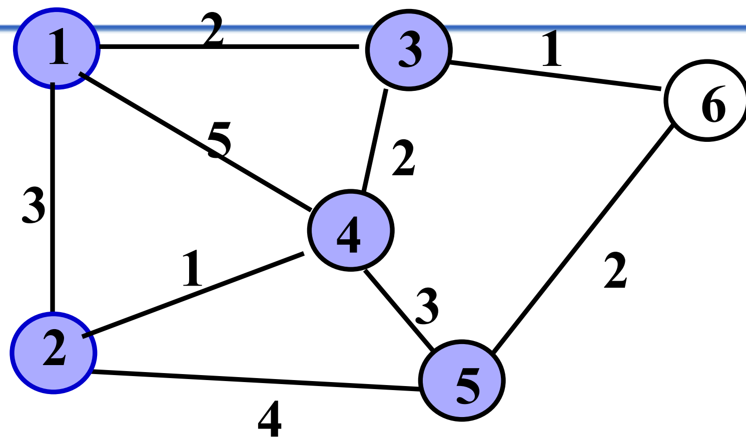
$D^{(4)}$

i / j	1	2	3	4	5	6
1	0	3	2	4	7	3
2	3	0	3	1	4	4
3	2	3	0	2	5	1
4	4	1	2	0	3	3
5	7	4	5	3	0	2
6	3	4	1	3	2	0

$P^{(4)}$

I / j	1	2	3	4	5	6
1	Nil	1	1	2	2	3
2	2	Nil	4	2	2	3
3	3	4	Nil	3	4	3
4	2	4	4	Nil	4	3
5	2	5	4	5	Nil	5
6	4	1	6	3	6	Nil

Floyd-warshall algorithm



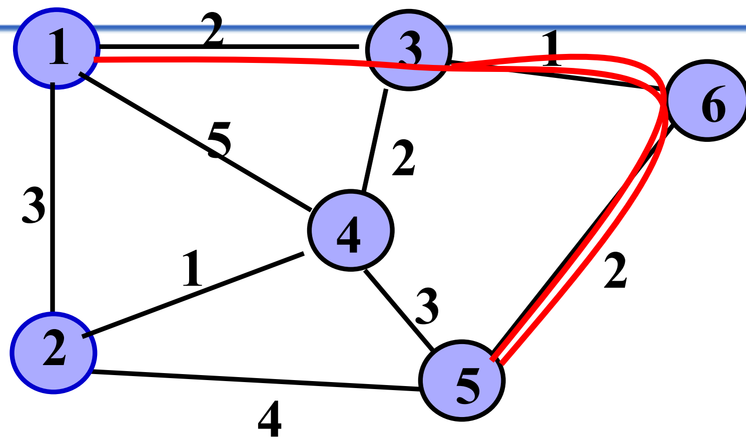
$D^{(4)}$

i / j	1	2	3	4	5	6
1	0	3	2	4	7	3
2	3	0	3	1	4	4
3	2	3	0	2	5	1
4	4	1	2	0	3	3
5	7	4	5	3	0	2
6	3	4	1	3	2	0

$P^{(4)}$

I / j	1	2	3	4	5	6
1	Nil	1	1	2	2	3
2	2	Nil	4	2	2	3
3	3	4	Nil	3	4	3
4	2	4	4	Nil	4	3
5	2	5	4	5	Nil	5
6	4	1	6	3	6	Nil

Floyd-warshall algorithm



$D^{(5)}$

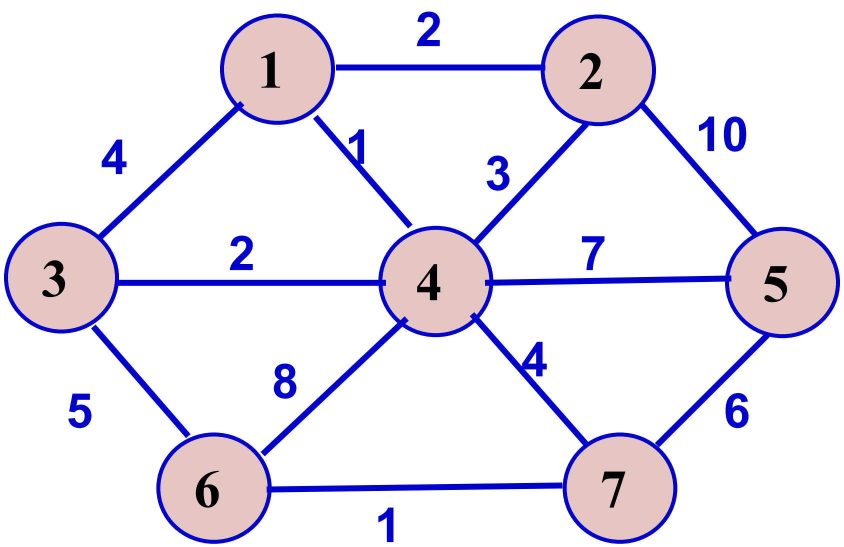
i / j	1	2	3	4	5	6
1	0	3	2	4	5	3
2	3	0	3	1	4	4
3	2	3	0	2	3	1
4	4	1	2	0	3	3
5	5	4	3	3	0	2
6	3	4	1	3	2	0

$P^{(5)}$

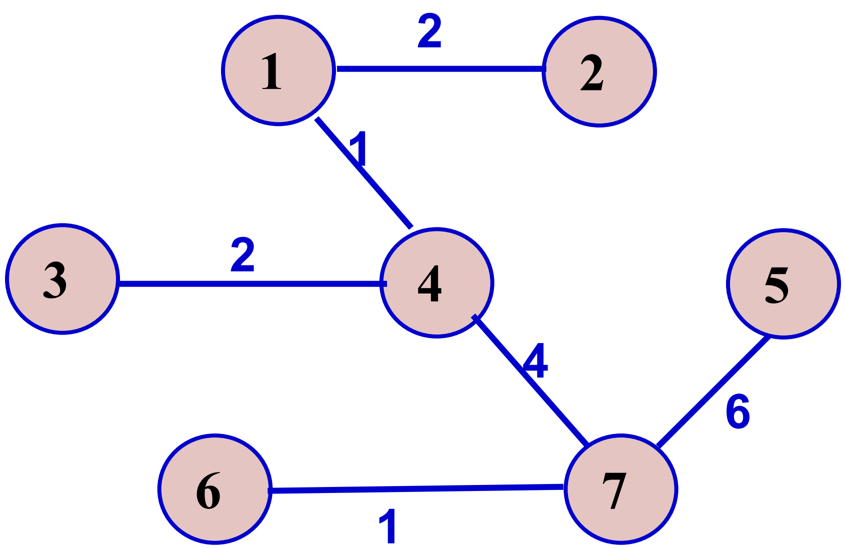
I / j	1	2	3	4	5	6
1	Nil	1	1	2	6	3
2	2	Nil	4	2	2	3
3	3	4	Nil	3	6	3
4	2	4	4	Nil	4	3
5	3	5	6	5	Nil	5
6	4	1	6	3	6	Nil

Minimum Spanning Tree (MST)

- *A* minimum spanning tree of an undirected graph G is a tree formed from graph edges that connects all the vertices of G at lowest total cost.
- A minimum spanning tree exists if and only if G is connected



A graph G



MST of G

Prim Algorithm

Connect all vertices with the **minimum total edge weight**, without forming any cycles.

Start with any vertex (this becomes the root of the MST).
Maintain two sets:

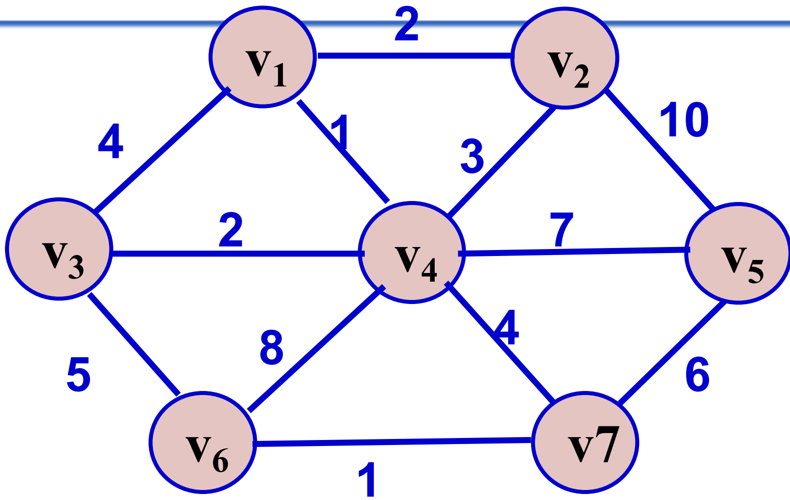
Tree vertices (T): already included in the MST.
Non-tree vertices: not yet in the MST.

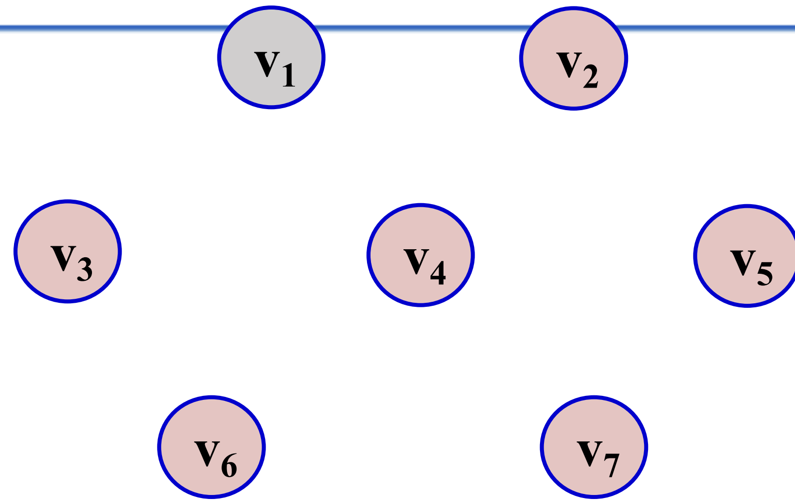
Prim Algorithm

- One way to compute a minimum spanning tree is to grow the tree in successive stages.
- In each stage,
  1. Pick a node as the root,
  2. add an edge, and
  3. the associated vertex, to the tree.
- At any point in the algorithm, we have
  - a set of vertices that have already been included in the tree;
  - the rest of the vertices are not in the tree yet
- At each stage, the algorithm finds
  - a new vertex to add to the tree by choosing the edge **(*u*, *v*)**
  - such that the cost of **(*u*, *v*)** is the smallest among all edges where ***u*** is in the tree and v is not.
- Same as Dijkstra's algo?

Prim Algorithm

v_1	v_4	v_2	v_3	v_7	v_6	v_5

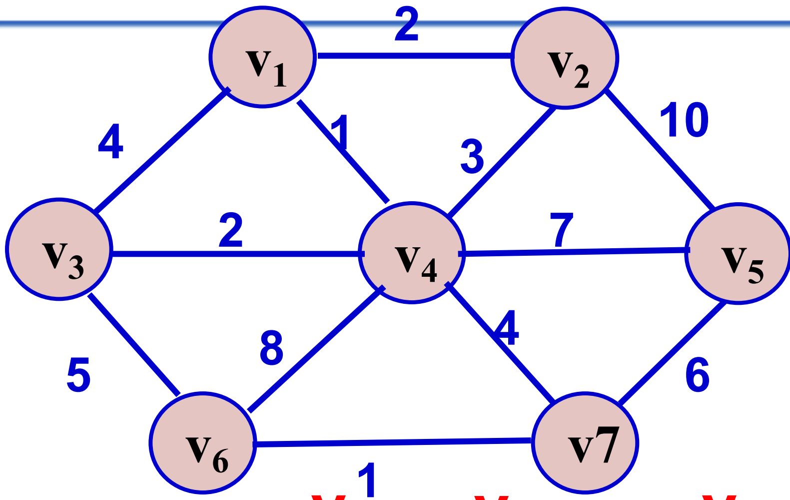


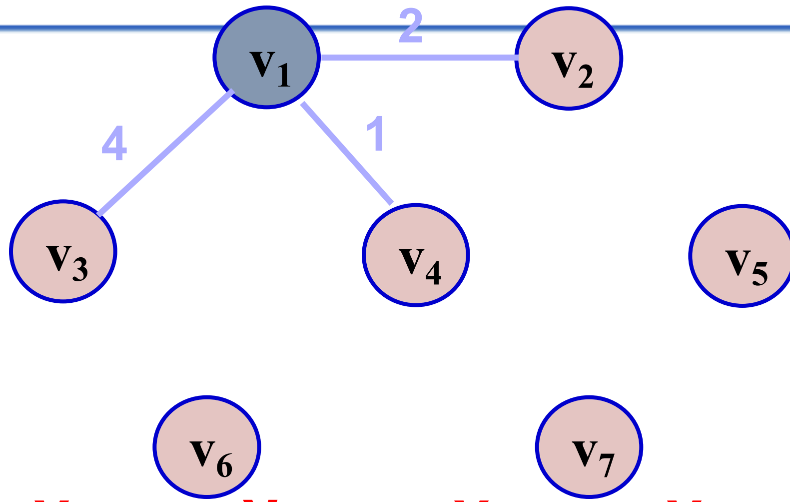


	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v	D_1 , p_1	D_2 , p_2	D_3 , p_3	D_4 , p_4	D_5 , p_5	D_6 , p_6	D_7 , p_7

Prim Algorithm

V_4	V_2	V_3	V_7	V_6	V_5

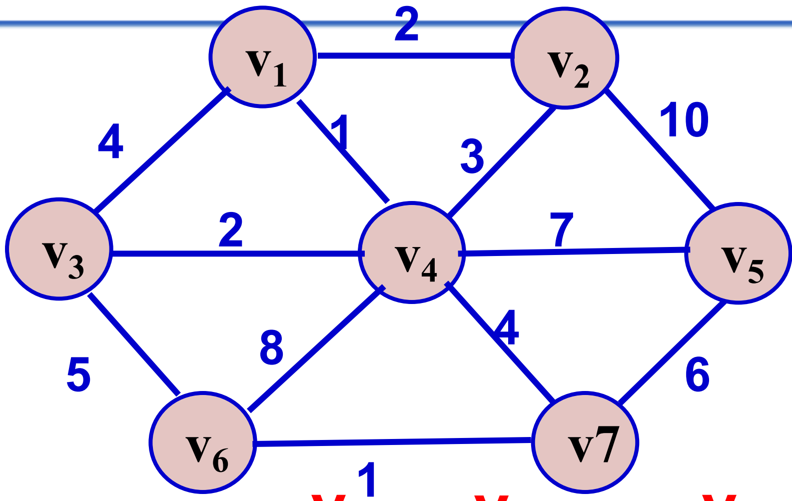


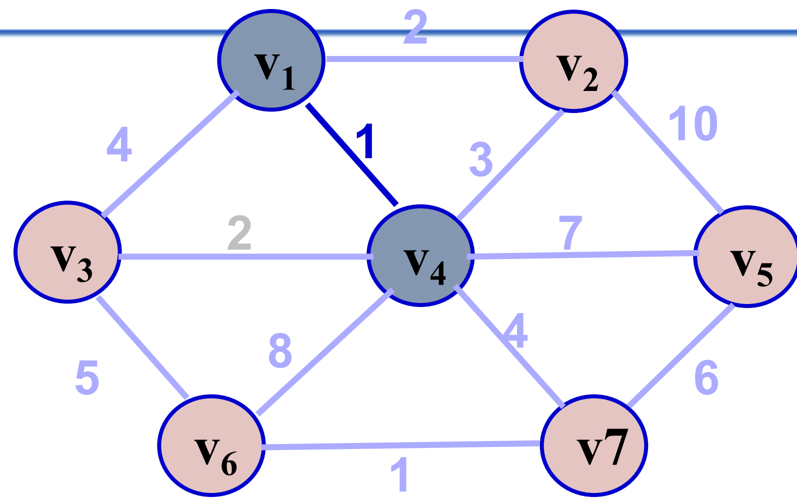


v	v_1 D_1 , p_1	v_2 D_2 , p_2	v_3 D_3 , p_3	v_4 D_4 , p_4	v_5 D_5 , p_5	v_6 D_6 , p_6	v_7 D_7 , p_7
1	0 , 0	2 , v_1	4 , v_1	1 , v_1	∞ , 0	∞ , 0	∞ , 0

Prim Algorithm

V_2	V_3	V_7	V_6	V_5

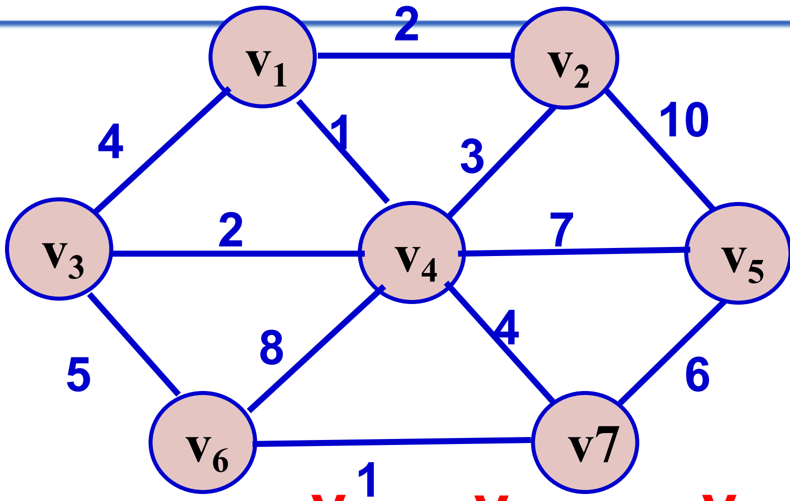


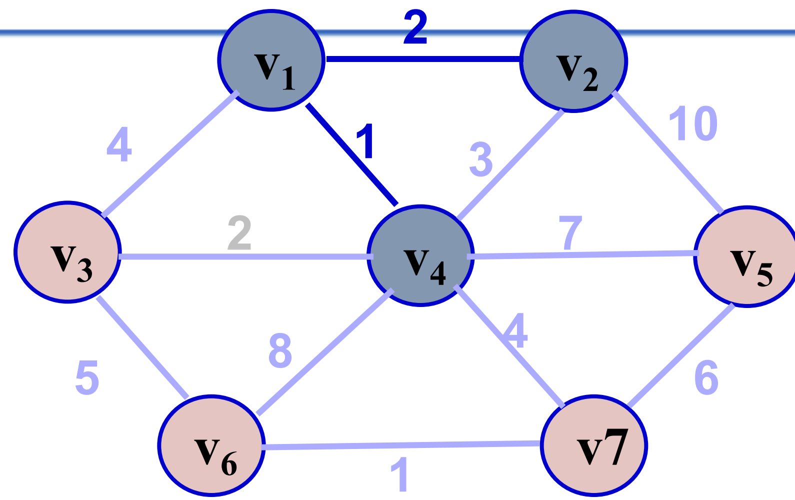


	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v	D_1 , p_1	D_2 , p_2	D_3 , p_3	D_4 , p_4	D_5 , p_5	D_6 , p_6	D_7 , p_7
1	0 , 0	2 , v_1	4 , v_1	1 , v_1	∞ , 0	∞ , 0	∞ , 0
1,4	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4

Prim Algorithm

V_3	V_7	V_6	V_5

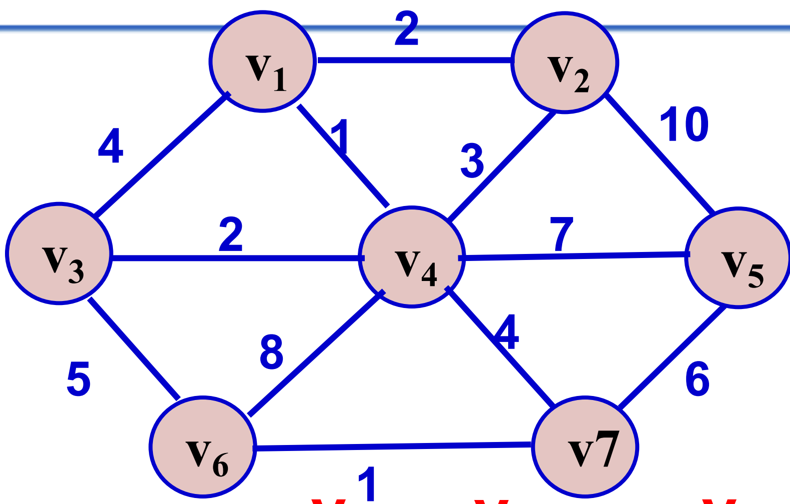


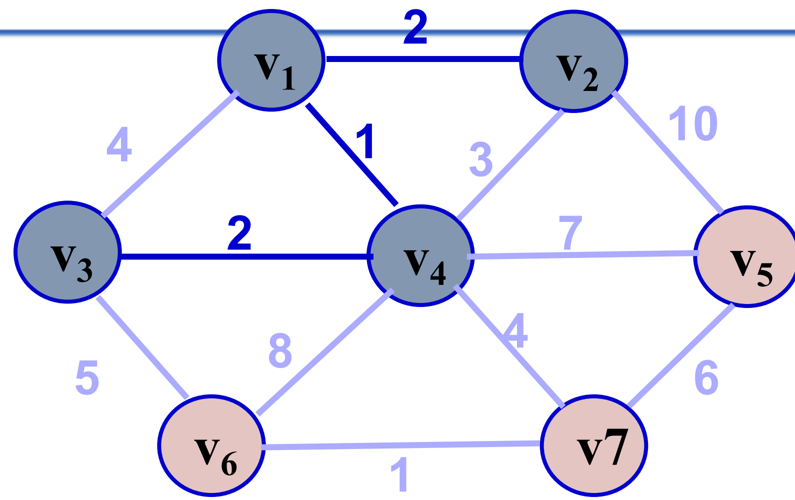


	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v	D_1 , p_1	D_2 , p_2	D_3 , p_3	D_4 , p_4	D_5 , p_5	D_6 , p_6	D_7 , p_7
1	0 , 0	2 , v_1	4 , v_1	1 , v_1	∞ , 0	∞ , 0	∞ , 0
1,4	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4

Prim Algorithm

V_7	V_6	V_5

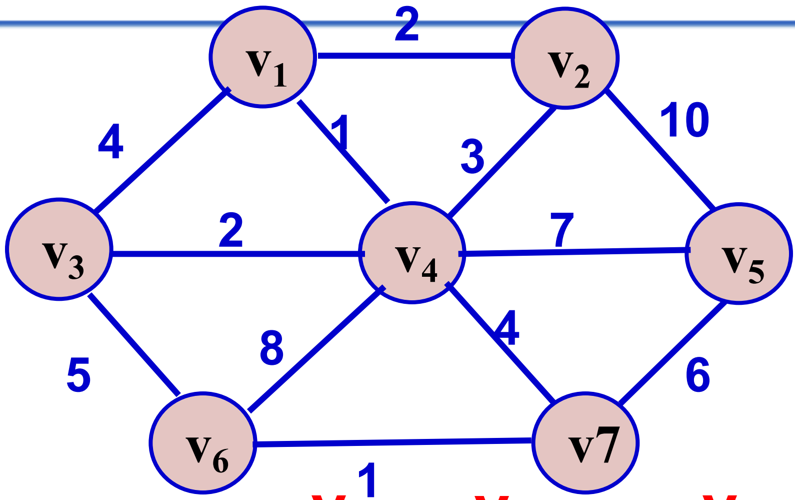


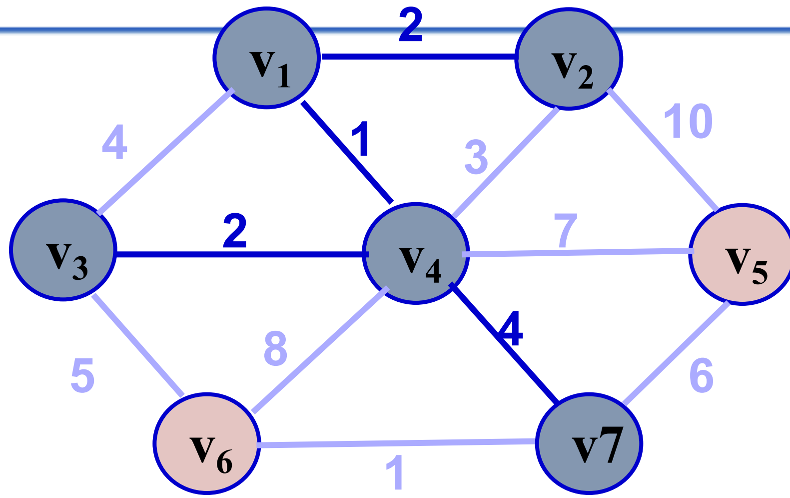


	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v	D_1 , p_1	D_2 , p_2	D_3 , p_3	D_4 , p_4	D_5 , p_5	D_6 , p_6	D_7 , p_7
1	0 , 0	2 , v_1	4 , v_1	1 , v_1	∞ , 0	∞ , 0	∞ , 0
1,4	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2,3	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	5 , v_3	4 , v_4

Prim Algorithm

V_6	V_5

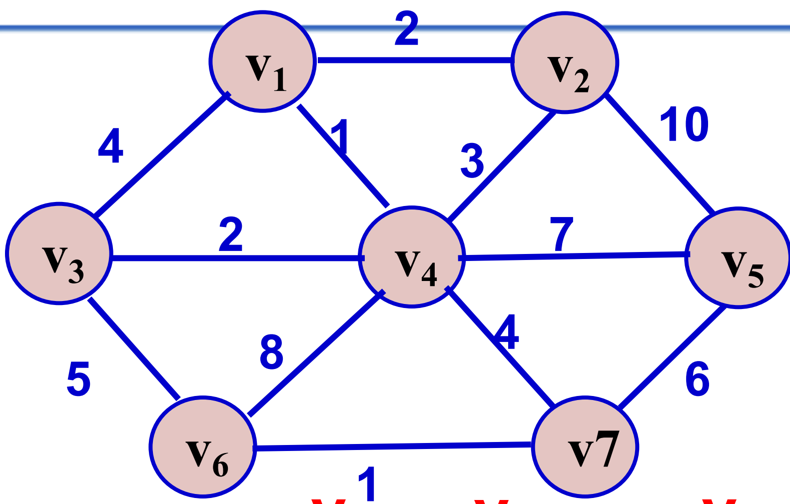


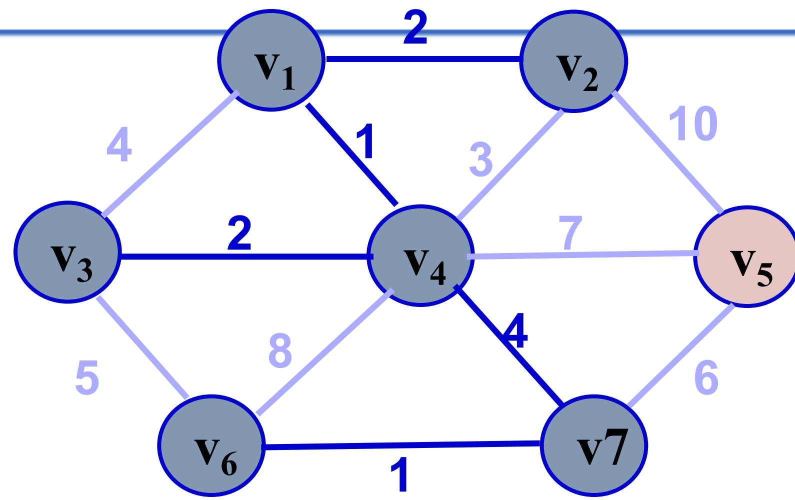


	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v	D_1 , p_1	D_2 , p_2	D_3 , p_3	D_4 , p_4	D_5 , p_5	D_6 , p_6	D_7 , p_7
1	0 , 0	2 , v_1	4 , v_1	1 , v_1	∞ , 0	∞ , 0	∞ , 0
1,4	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2,3	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	5 , v_3	4 , v_4
1,4,2,3,7	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4

Prim Algorithm

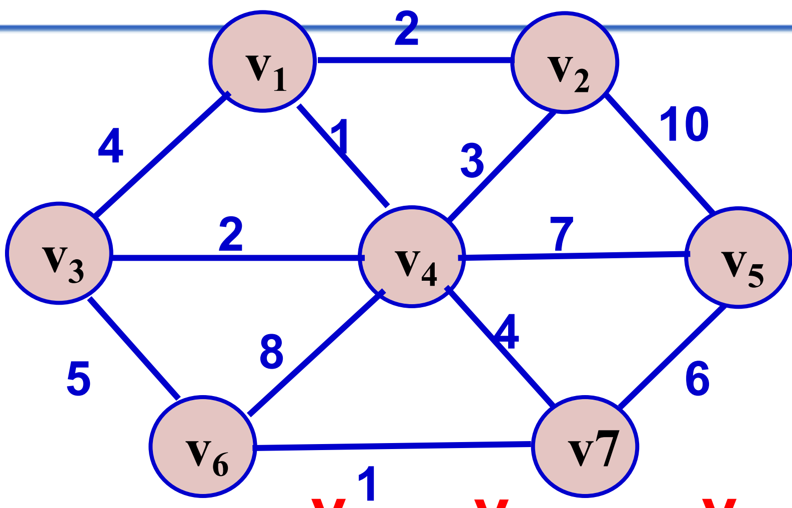
V_5

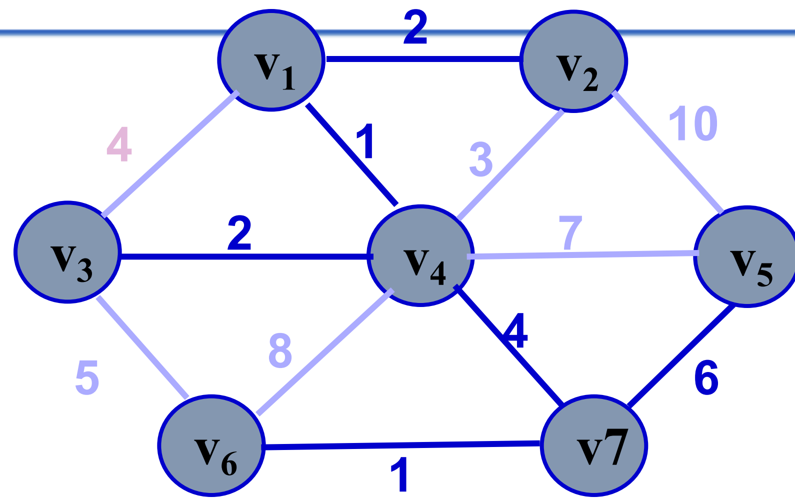




	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v	D_1 , p_1	D_2 , p_2	D_3 , p_3	D_4 , p_4	D_5 , p_5	D_6 , p_6	D_7 , p_7
1	0 , 0	2 , v_1	4 , v_1	1 , v_1	∞ , 0	∞ , 0	∞ , 0
1,4	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2,3	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	5 , v_3	4 , v_4
1,4,2,3,7	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4
1,4,2,3,7,6	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4

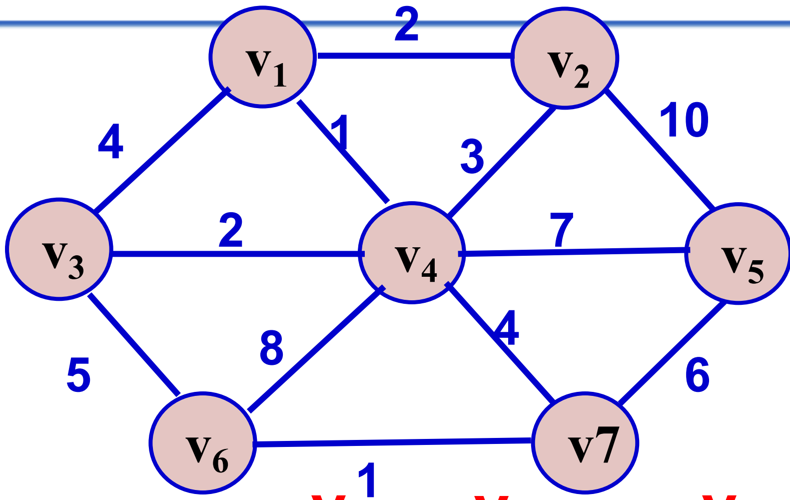
Prim Algorithm

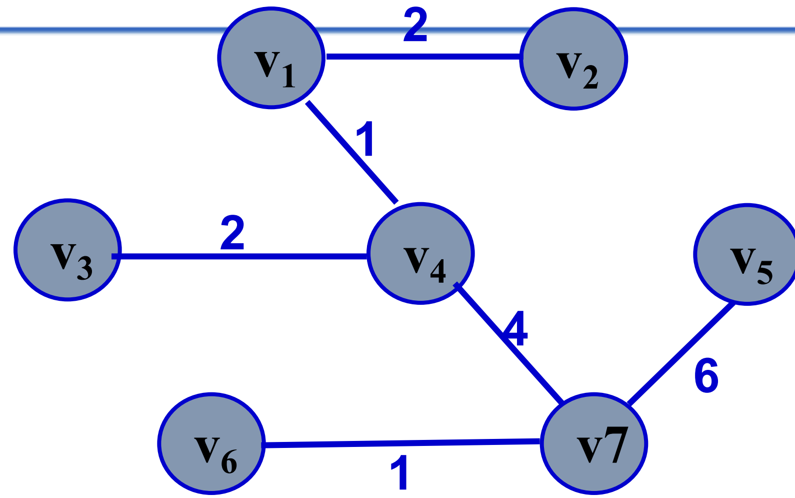




	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v	D_1 , p_1	D_2 , p_2	D_3 , p_3	D_4 , p_4	D_5 , p_5	D_6 , p_6	D_7 , p_7
1	0 , 0	2 , v_1	4 , v_1	1 , v_1	∞ , 0	∞ , 0	∞ , 0
1,4	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2,3	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	5 , v_3	4 , v_4
1,4,2,3,7	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4
1,4,2,3,7,6	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4
1 to 7	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4

Prim Algorithm





	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v	D_1 , p_1	D_2 , p_2	D_3 , p_3	D_4 , p_4	D_5 , p_5	D_6 , p_6	D_7 , p_7
1	0 , 0	2 , v_1	4 , v_1	1 , v_1	∞ , 0	∞ , 0	∞ , 0
1,4	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	8 , v_4	4 , v_4
1,4,2,3	0 , 0	2 , v_1	2 , v_4	1 , v_1	7 , v_4	5 , v_3	4 , v_4
1,4,2,3,7	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4
1,4,2,3,7,6	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4
1 to 7	0 , 0	2 , v_1	2 , v_4	1 , v_1	6 , v_7	1 , v_7	4 , v_4

Kruskal Algorithm

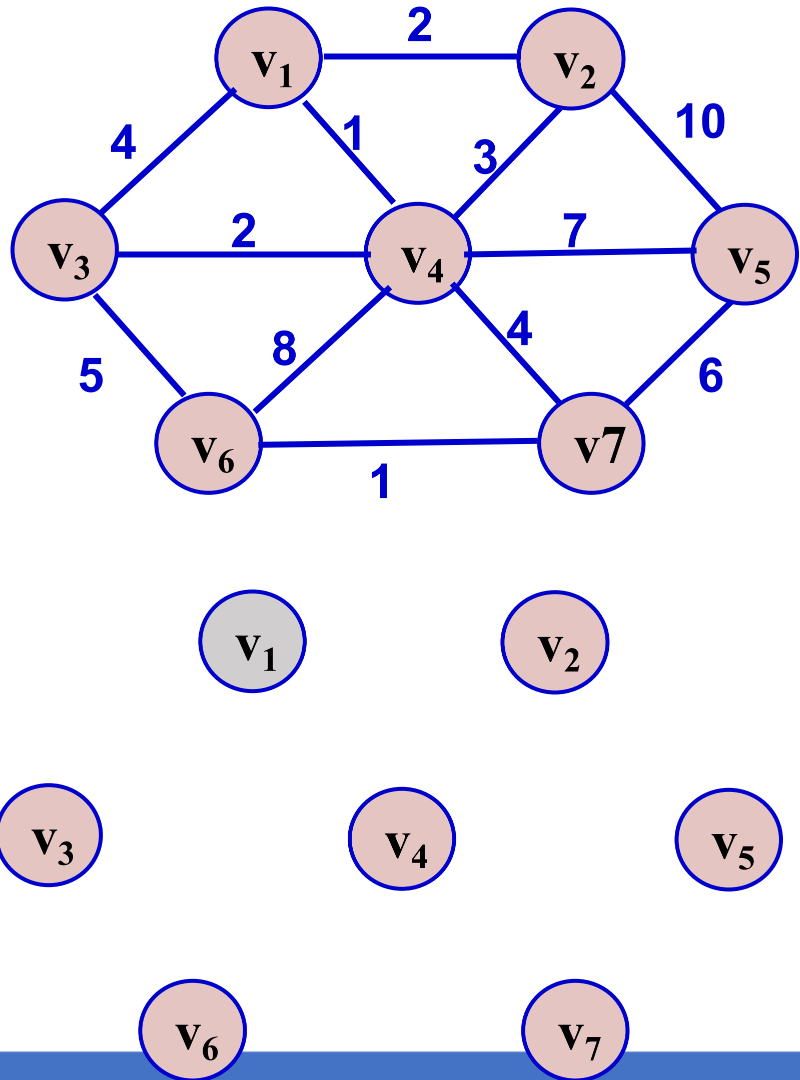
MST-Kruskal (G,w)

```
F = { }
For each vertex v ∈ V[G]
  MakeSet(v)
sort the edges(E) in increasing order by
weight w
for each edge (u,v) ∈ E
  if FindSet(u) ≠ FindSet(v)
      F = F U {(u,v)}
        UNION(u,v)
```

Kruskal Algorithm

- MakeSet(***v***): Create a new set whose only member is pointed to by ***v***. Note that for this operation ***v*** must already be in a set.
- FindSet(v): Returns a pointer to the set containing ***v***.
- UNION(**u, v**): Unites the dynamic sets that contain ***u*** and ***v*** into a new set that is union of these two sets.

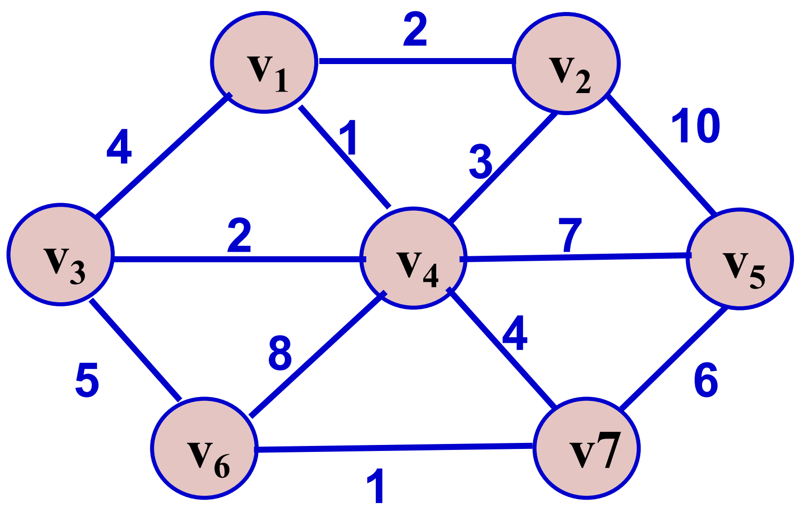
Kruskal’s Algorithm



SETS

$\{ v_1 \}$,
$\{ v_2 \}$,
$\{ v_3 \}$,
$\{ v_4 \}$,
$\{ v_5 \}$,
$\{ v_6 \}$,
$\{ v_7 \}$

Kruskal's Algorithm



SETS

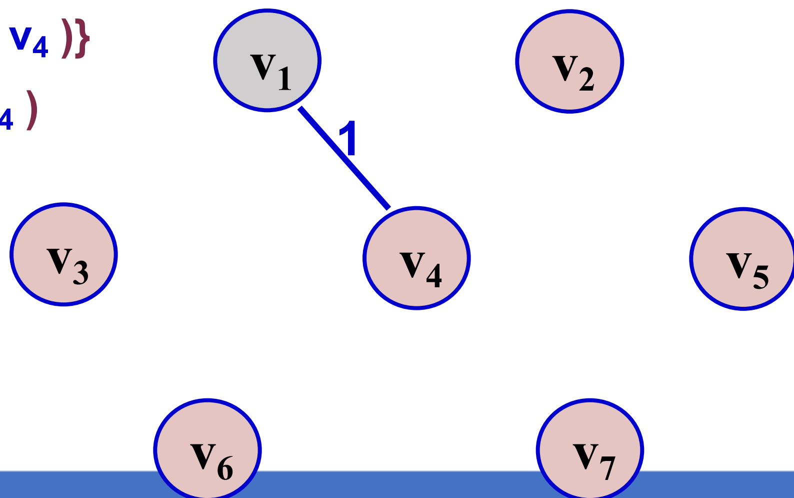
$\{ v_1 , v_4 \}$,
$\{ v_2 \}$,
$\{ v_3 \}$,
$\{ v_5 \}$,
$\{ v_6 \}$,
$\{ v_7 \}$

Min Edge = (v_1, v_4)

FindSet(v_1) $\neq$ FindSet(v_4)

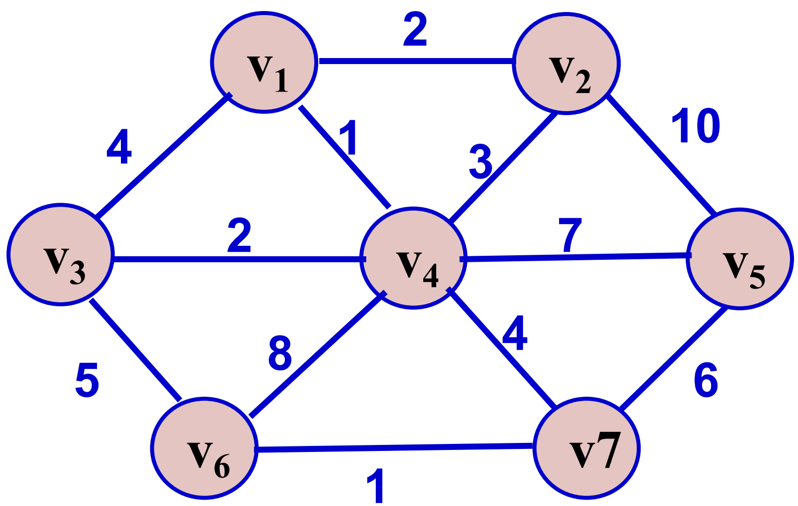
$F = F \cup \{(v_1 , v_4)\}$

UNION(v_1 , v_4)



$F = \{ (v_1 , v_4) \}$

Kruskal's Algorithm



SETS

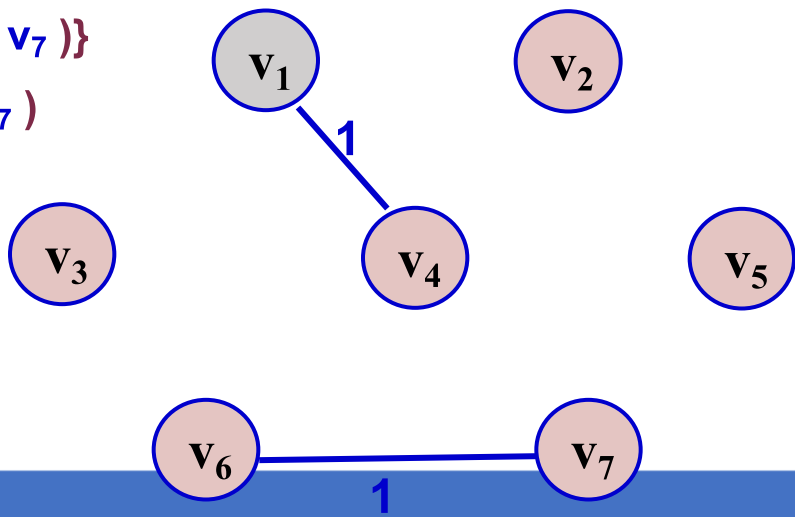
$\{ v_1 , v_4 \}$,
$\{ v_2 \}$,
$\{ v_3 \}$,
$\{ v_5 \}$,
$\{ v_6 , v_7 \}$

Min Edge = (v_6, v_7)

FindSet(v_6) $\neq$ FindSet(v_7)

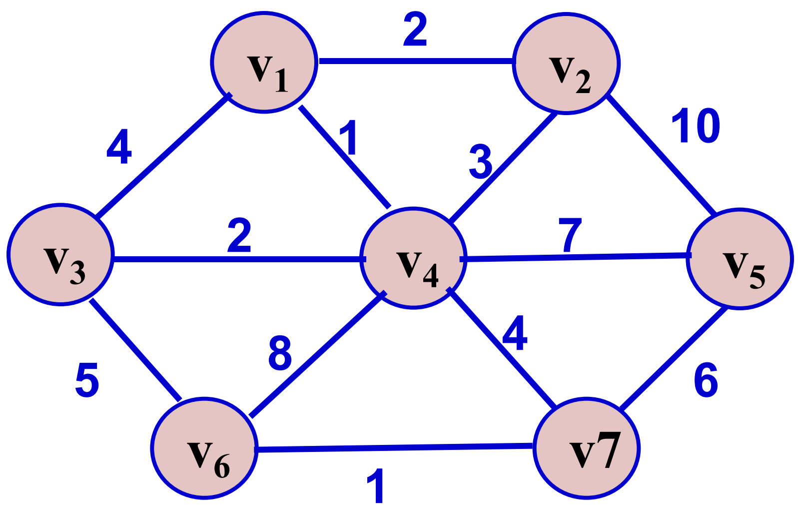
$F = F \cup \{(v_6 , v_7)\}$

UNION(v_6 , v_7)



$F = \{ (v_1 , v_4) ,$
$(v_6 , v_7) \}$

Kruskal’s Algorithm



SETS

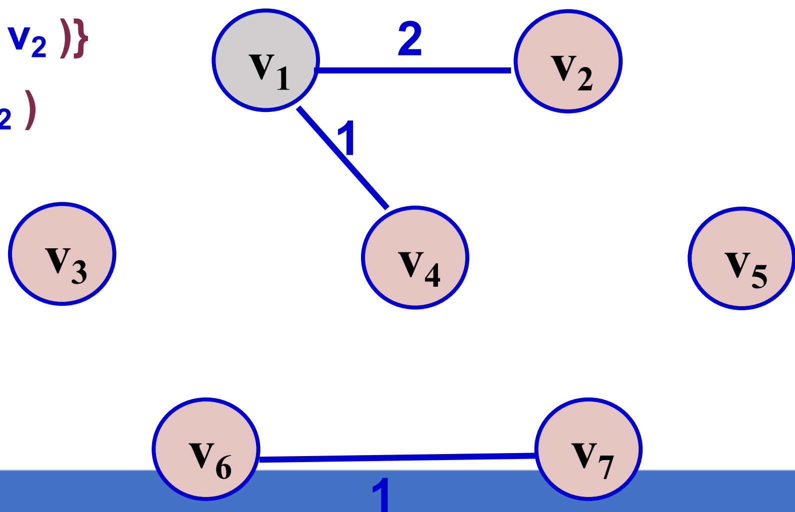
$\{v_1, v_2, v_4\}$,
$\{v_3\}$,
$\{v_5\}$,
$\{v_6, v_7\}$

Min Edge = (v_1, v_2)

FindSet(v_1) $\neq$ FindSet(v_2)

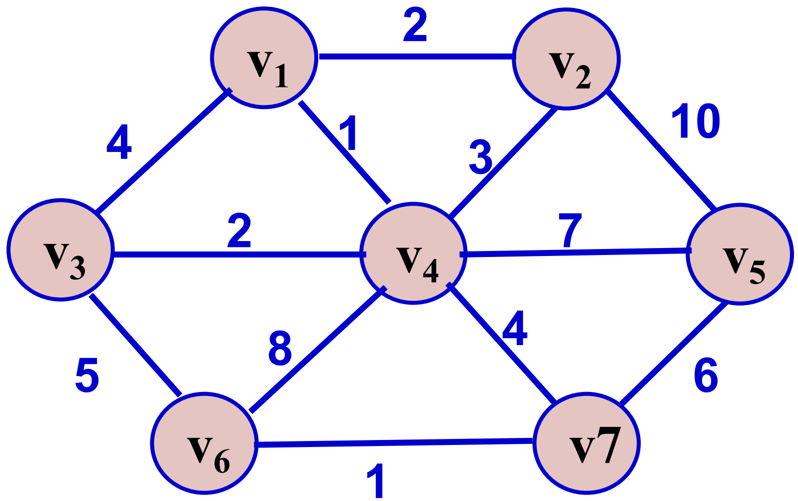
$F = F \cup \{(v_1, v_2)\}$

UNION(v_1 , v_2)



$F = \{ (v_1, v_4),$
$(v_6, v_7),$
$(v_1, v_2) \}$

Kruskal's Algorithm



SETS

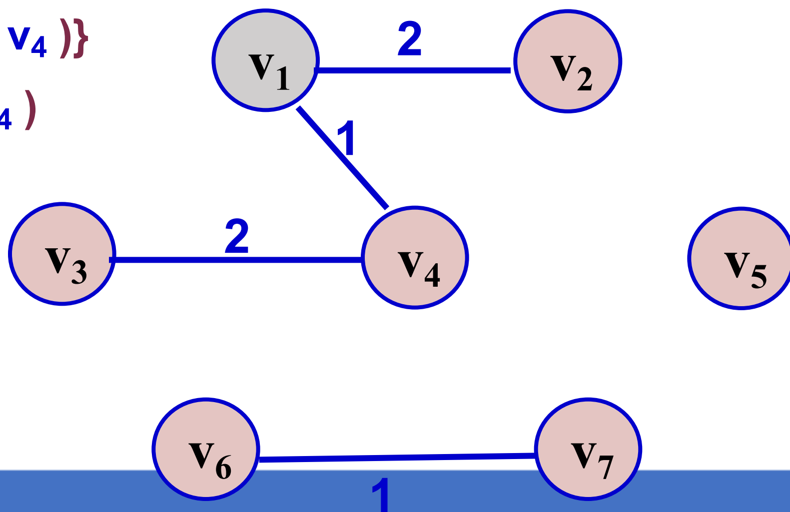
$\{v_1 , v_2 , v_3 , v_4\}$,
$\{ v_5 \}$,
$\{ v_6 , v_7 \}$

Min Edge = (v_3, v_4)

FindSet(v_3) $\neq$ FindSet(v_4)

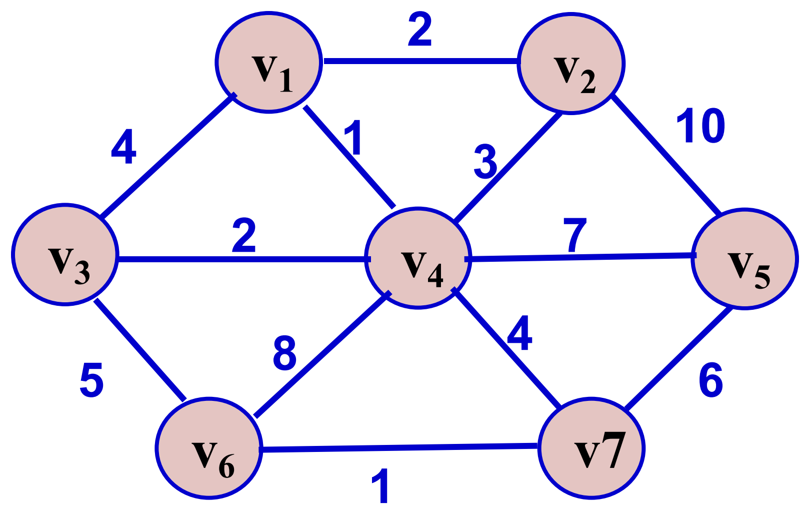
$F = F \cup \{(v_3 , v_4)\}$

UNION(v_3 , v_4)



$F = \{ (v_1 , v_4) ,$
$(v_6 , v_7) ,$
$(v_1 , v_2) ,$
$(v_3 , v_4) \}$

Kruskal's Algorithm



SETS

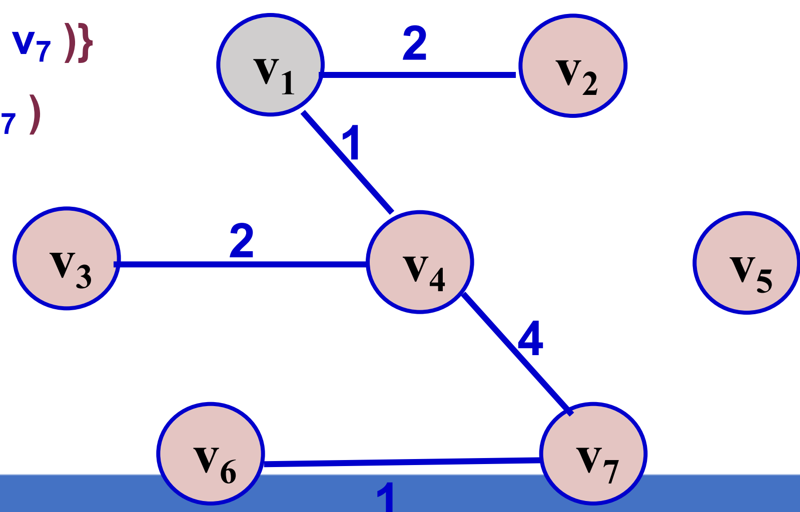
$\{v_1, v_2, v_3, v_4, v_6, v_7\}$,
$\{v_5\}$

Min Edge = (v_4, v_7)

FindSet(v_4) $\neq$ FindSet(v_7)

F = *F* U {(v_4 , v_7)}

UNION(v_4 , v_7)

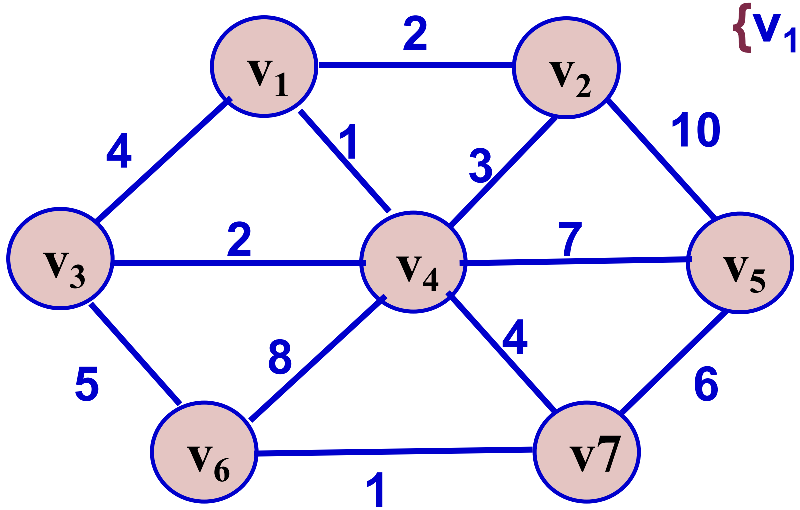


F ={ (v_1 , v_4) ,
(v_6 , v_7) ,
(v_1 , v_2) ,
(v_3 , v_4) ,
(v_4 , v_7) }

Kruskal’s Algorithm

FINAL SET

$\{v_1 , v_2 , v_3 , v_4 , v_5 , v_6 , v_7\}$,

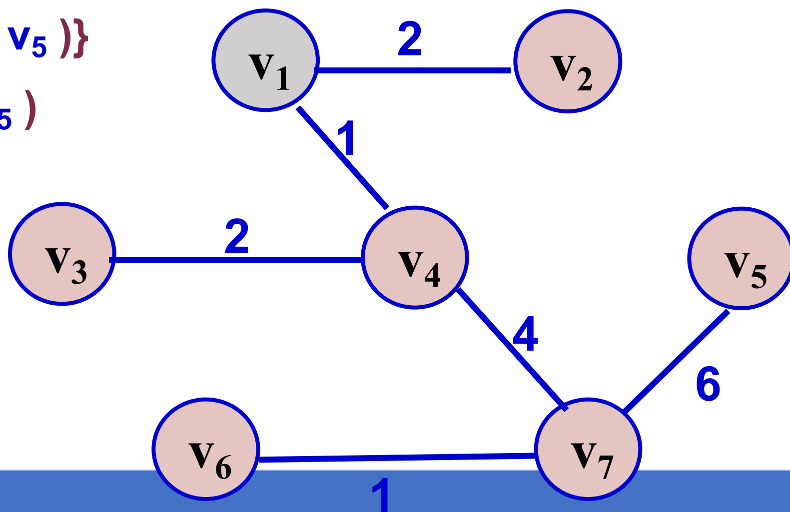


Min Edge = (v_7, v_5)

FindSet(v_7) $\neq$ FindSet(v_5)

$F = F \cup \{(v_7 , v_5)\}$

UNION(v_7 , v_5)



$F =\{ (v_1 , v_4) ,$
$(v_6 , v_7) ,$
$(v_1 , v_2) ,$
$(v_3 , v_4) ,$
$(v_4 , v_7) ,$
$(v_7 , v_5) \}$

Questions?

zahmaad.github.io