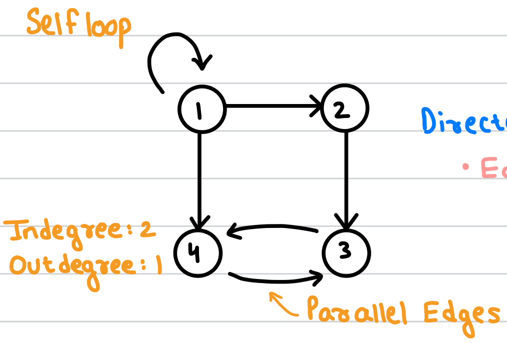
# GRAPHS

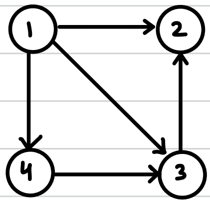
$G = (V, E)$

$V$ = Set of vertices

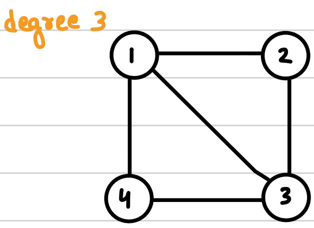
$E$ = Set of edges



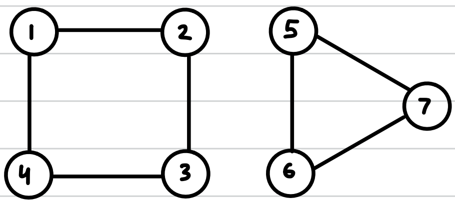

**Directed Graph**
- Edges are having directions



**Simple Diagraph**
- Without self loop
- Without parallel edges

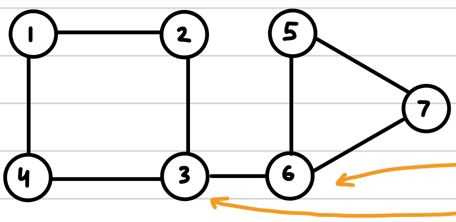



**Graph / Non-directed Graph**
- Undirected Edges



2 components

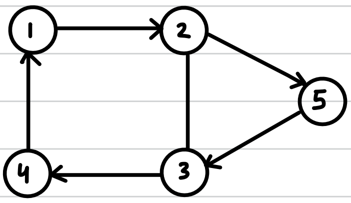
**Non-connected Graph**



Connected components

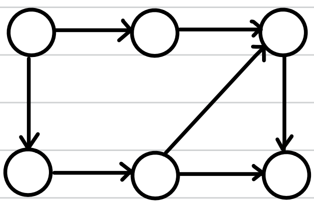
**Connected Graph**

Articulation Points
- Those vertices whose removal will divide the graph into multiple components



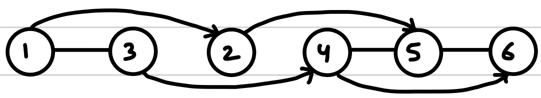

**Strongly Connected Graph** (Directed Graph)

- All vertices can be reached from any vertex
- There is a path between every pair of vertices
- Path is set of vertices which are connecting pair of vertices
- Cycle is a circular path that is starting from same vertex and ending at same vertex



**Directed Acyclic Graph (DAG)**

- Directed Graph
- with no cycles
- These can be arranged linearly such that edges are going in only forward direction

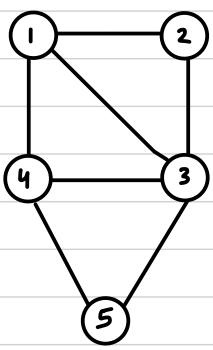



Topological Ordering

## REPRESENTATION OF UNDIRECTED GRAPH

(1) Adjacency Matrix
(2) Adjacency List
(3) Compact List

### ADJACENCY MATRIX




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 | 1 |
| 4 | 1 | 0 | 1 | 0 | 1 |
| 5 | 0 | 0 | 1 | 1 | 0 |

5 x 5

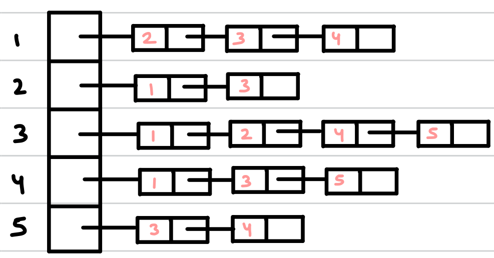
$|V| = n = 5$

$|E| = e = 7$

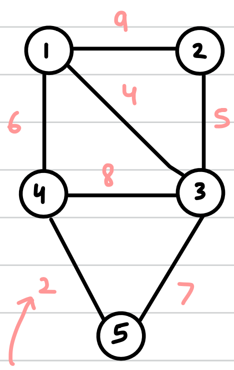
$(i, j)$

$A[i][j] = 1$

### ADJACENCY LIST



## COST ADJACENCY MATRIX (Representation of weight)




weight of edge

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 9 | 4 | 6 | 0 |
| 2 | 9 | 0 | 5 | 0 | 0 |
| 3 | 4 | 5 | 0 | 8 | 7 |
| 4 | 6 | 0 | 8 | 0 | 2 |
| 5 | 0 | 0 | 7 | 2 | 0 |

5 x 5

## COST ADJACENCY LIST

Add another area to node for weight of edge

## COMPACT LIST

Ending point of last vertex

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ~~ ~~ | 7 | 10 | 12 | 16 | 19 | 21 | 2 | 3 | 4 | 1 | 3 | 1 | 2 | 3 | 4 | 1 | 3 | 5 | 3 | 4 |
| | Vertices | | | | | | 1 | | | 2 | | 3 | | | | 4 | | | 5 | |

Starting indexes

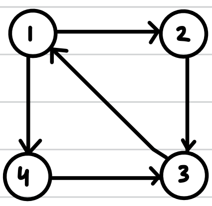
$|V| + 2|E| + 1$

$5 + 2 \times 7 + 1 = 20$

$20 + 1 = 21$

## REPRESENTATION OF DIRECTED GRAPH

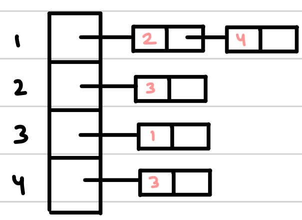
### ADJACENCY MATRIX



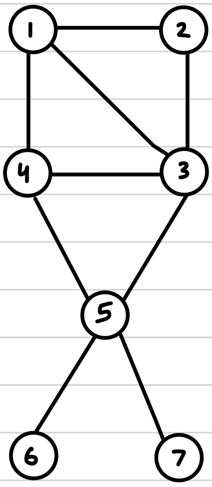

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |

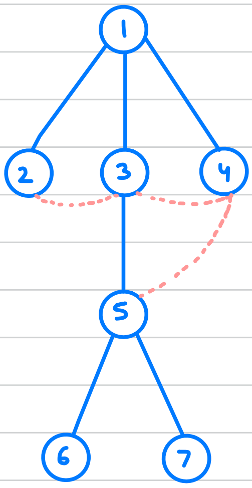
4 x 4

### ADJACENCY LIST

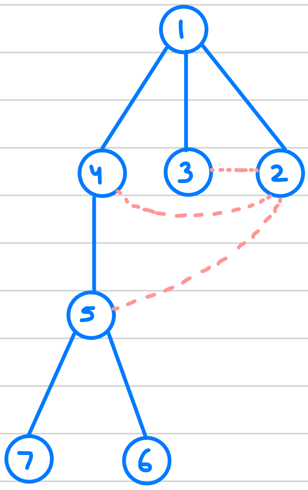


# BREADTH FIRST SEARCH (BFS)




TREE 1




TREE 2

Multiple trees can be generated from the same graph

- We can start from any vertex
- Each vertex should be explored fully
- Dotted pink lines represent edges that make a complete cycle called cross edges
- These trees are called BFS Spanning Trees

BFS TREE 1 : 1, 2, 3, 4, 5, 6, 7 // Same as Level order Traversal in trees
BFS TREE 2 : 1, 4, 3, 2, 5, 7, 6

Starting vertex

$O(n^2)$

```
void BFS (int i)
{
    int u, v;
    printf("%d", i);
    visited[i] = 1;
    enqueue (q, i);

    while ( ! isEmpty (q))
    {
        u = dequeue (q);      Take out vertex from queue

        for( v = 1; v <= n; v++)
        {
            if ( A[u][v] == 1 && visited[v] == 0)
            {
                printf(" %d", v);
                visited [v] = 1;
                enqueue (q, v);
            }
        }
    }
}
```

For scanning row of matrix

Should not be visited

| A | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | |
| 1 | | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 4 | | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 5 | | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 7 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

QUEUE q

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |

visited

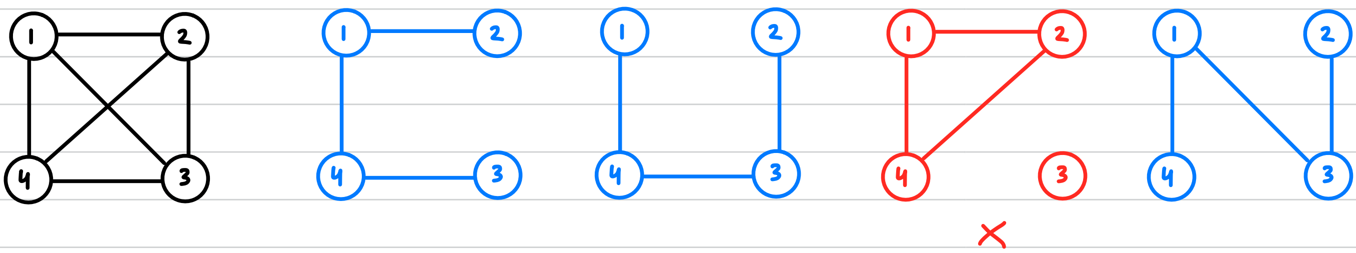
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

## DEPTH FIRST SEARCH (DFS)

```
void DFS (int u)
{
    int v;

    if ( visited[u] == 0)
    {
        printf("%d", u);
        visited[u] = 1;

        for( v = 1; v <= n; v++)
        {
            if ( A[u][v] == 1 && visited[v] != 0)
                DFS(v);
        }
    }
}
```

## SPANNING TREES

Spanning tree is a sub graph of a graph having all vertices of a graph and |V|-1 edges and there should not be any cycle.



$|V| = 4$

$|E| = 6$

$^{|E|}C_{|V|-1} = {}^{6}C_{3}$ ways or number of spanning trees

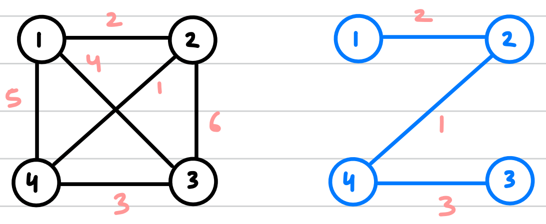
but not include trees forming cycles

$^{|E|}C_{|V|-1} - \text{cycles} = {}^{6}C_{3} - 4$ // In this example

$= 16$

## MINIMUM COST SPANNING TREE

If weights are added to edges of graph, then the spanning tree with minimum cost of weights is called minimum cost spanning tree.



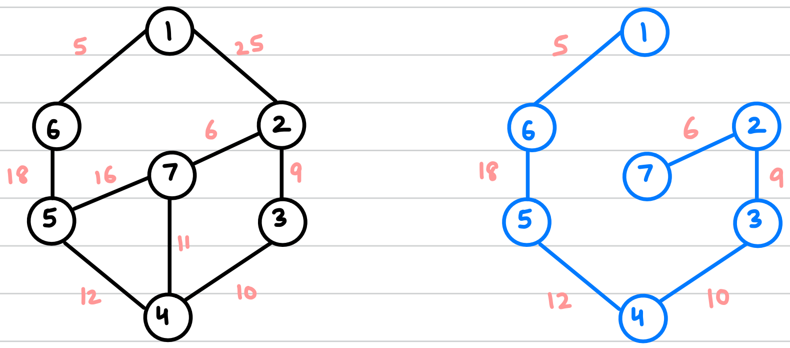
= 6

## PRIM'S MINIMUM COST SPANNING TREE

$(|V|-1)\,|E|$

$ne = n \times n$

$O(n^2)$

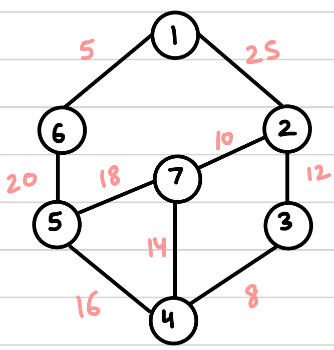


**STEP 1 :** Select the edge with least weight along with its vertices

**STEP 2 :** Then compare the edges joined with the vertices and select the one with least weight along with its vertex

**STEP 3 :** Repeat step 2 until spanning tree is obtained

## PROGRAM



| COST | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | - | - | - | - | - | - | - | - |
| 1 | - | - | 25 | - | - | - | 5 | - |
| 2 | - | 25 | - | 12 | - | - | - | 10 |
| 3 | - | - | 12 | - | 8 | - | - | - |
| 4 | - | - | - | 8 | - | 16 | - | 14 |
| 5 | - | - | - | - | 16 | - | 20 | 18 |
| 6 | - | 5 | - | - | - | 20 | - | - |
| 7 | - | - | 10 | - | 14 | 18 | - | - |

near

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| ~~-~~ | - | - | - | - | - | - | - |

t

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | | | | | | |
| 1 | | | | | | |

// Only either of lower or upper triangular matrix will be sufficient

Largest possible integer value

```
#define I 32767
```

Initialize as {I}

```
int cost[8][8], near[8], t[2][6];
```

Initialize as the given matrix (Replace '-' with 'I')

INITIAL PROCEDURE

```
void main()
{
    int i, j, k, u, v, n=7, min = I;

    for (i=1; i<=n; i++)
    {
```

For accessing upper triangular matrix only

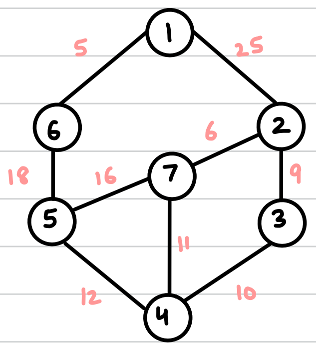
```
        for(j=i; j<=n; j++)
        {
            if(cost[i][j]<min)
            {
                min = cost[i][j];
                u=i, v=j;
            }
        }
    }

    t[0][0] = u;
    t[1][0] = v;

    near[u] = near[v] = 0;

    for (i=1; i<=n; i++)
    {
        if(near[i]!=0 && cost[i][u] < cost[i][v])
                near[i]=u;
        else
                near[i]=v;
    }
```

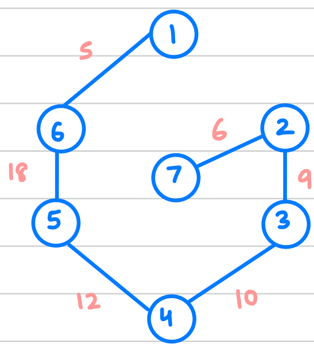
RECURSIVE PROCEDURE

```
    for (i=1; i<n-1; i++)
    {
        min = I;

        for(j=1; j<=n; j++)
        {
            if (near[j]!=0 && cost[j][near[j]]<min)
            {
```

```
                min = cost[j][near[j]],
                k = j;
            }
    }

    t[0][i] = k;
    t[1][i] = near[k];
    near[k] = 0;

    for (j=1; j<=n; j++)
    {
            if (near[j]!=0 && cost[j][k] < cost[j][new[j]])
                near[j] = k;
    }
  }

  Print t;
}
```

## KRUSKAL'S METHOD




**STEP 1 :** Select the edge with least weight along with its vertices

**STEP 2 :** Select the next minimum edge if it is not forming a cycle

## DISJOINT SUBSET

$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

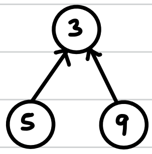
Consider its two subsets

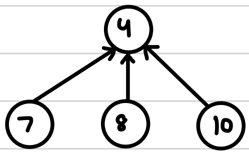
$A = \{3, 5, 9\}$

$B = \{4, 7, 8, 10\}$

$A \cap B = \phi$ // Disjoint subset

$A = \{3, 5, 9\}$ $B = \{4, 7, 8, 10\}$



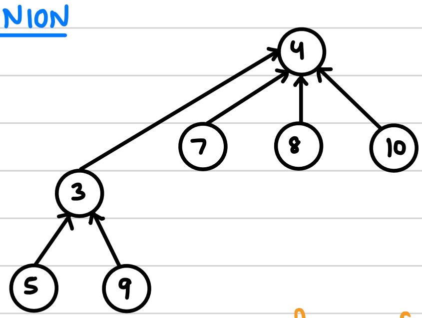

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | -3 | -4 | 3 | | 4 | 4 | 3 | 4 |

No of nodes in negative
(4 is having 4 nodes)

Rest are representing parent

Representation of set

(1) UNION




The parent with more number of nodes

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 4 | -7 | 3 | | 4 | 4 | 3 | 4 |

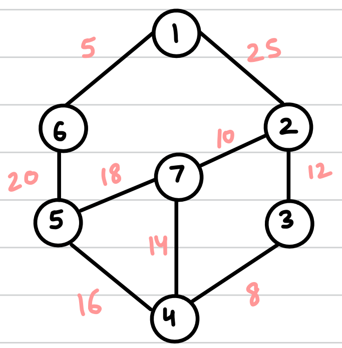
Parent of nodes

```
void Union (int u, int v)
{
    if ( s[u] < s[v] )
    {
        s[u] = s[u] + s[v];
        s[v] = u;
    }
    else
    {
        s[v] = s[u] + s[v];
        s[u] = v;
    }
}
```

(2) FIND

```
int Find (int u)
{
    int x = u;

    while ( s[x] > 0 )
        x = s[x];

    while ( u != x )
    {
        v = s[u];
        s[u] = x;
        u = v;
    }
}
```

Connecting vertices directly to parent



|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 5 | 5 |
| 1 | 2 | 6 | 3 | 7 | 4 | 5 | 7 | 6 | 7 |
| 2 | 25 | 5 | 12 | 10 | 8 | 16 | 14 | 20 | 18 |

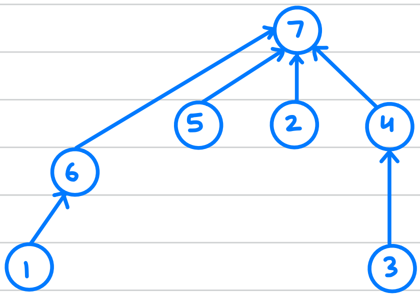
edges

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| x | 6 | 7 | 4 | 7 | 7 | -2 | -5 |

set

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

included



t

|  | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1 | 3 | 2 | 2 | 4 | 5 |
| 1 | 6 | 4 | 7 | 3 | 5 | 6 |

```
#define  I  32767

int  edges[3][9] = { { 1,1, 2,2, 3,4,4, 5,5},
                     { 2,6, 3,7, 4,5,7, 6,7},
                     { 25,5,12,10,18,16,14,20,18}};

int   set[8] = {-1};
int   included[9] = {0} , t[2][6];


void main()
{
      int  i=0,j,k,n=7,e=9,min,u,v;
                          No of vertices -1
      while (i < n-1)
      {
            min = I;
                               For finding minimum cost edge
            for(j=0;j<e;j++)
            {
                  if ( included[j]==0   && edges[2][j] <min)
                  {
                        min = edges[2][j];
                        k = j ;
                        u = edges[0][j];
                        v = edges[1][j];
                  }
            }
```

```
        if ( find (u) != find (v))        To check if cycle is
        {                                 forming or not
              t[0][i] = u ;
              t[1][i] = v ;
              union ( find (u) , find (v) ) ;
              i++ ;
        }

        included [k] = 1;
    }
}
```

## ASYMPTOTIC NOTATION

$$f(n) = \sum_{i=1}^{n} i \; - \; 1+2+3 \ldots\ldots +n \; = \; \frac{n(n+1)}{2} \; = \; O(n^2)$$ We can get exact value

$$f(n) = \sum_{i=1}^{n} i \times 2^i$$ Exact value is not possible that is simplified

So, we use asymptotic notations

| | | | | |
|---|---|---|---|---|
| Lower Bound | $\Omega$ | Omega | less than or equal to |
| Upper Bound | $O$ | Big O | Greater than or equal to |
| Tight Bound | $\theta$ | Theta | equal |