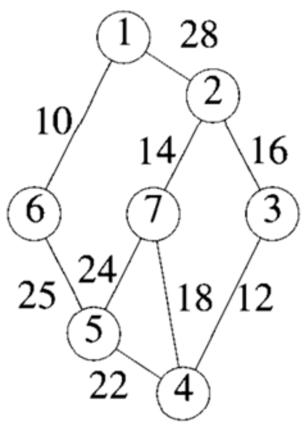
# Prim's Algorithm

# Prim's Algorithm

- Prim's algorithm starts with a null Spanning Tree(MST).
- At any given time, It adds a least cost edge(selection) to the spanning tree such that
  - the resulting spanning tree continues to be a tree (feasible:no cycles are formed by adding the new edge).
  - The cost of spanning tree continues to be minimum.

# Prim's Algorithm

- The outline of the algorithm is:
- Initially, add the least cost edge to the spanning tree.

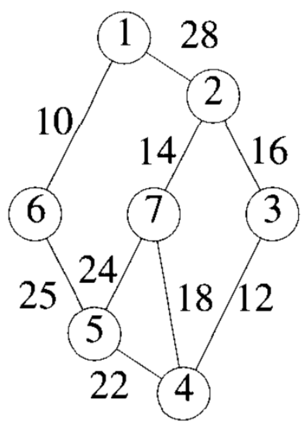


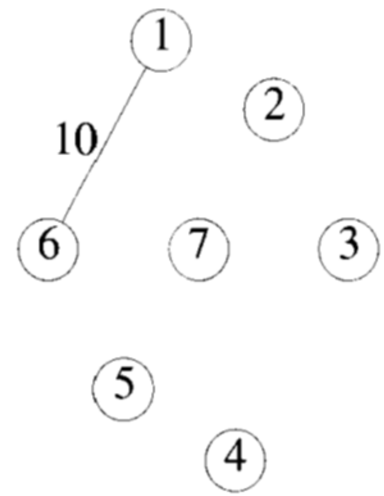
# Prim's Algorithm

- Prim's algorithm maintains two sets of vertices to accomplish this.
- One set contains tree vertices that are in the spanning tree.
- The other set contains the pending vertices yet to join the spanning tree.

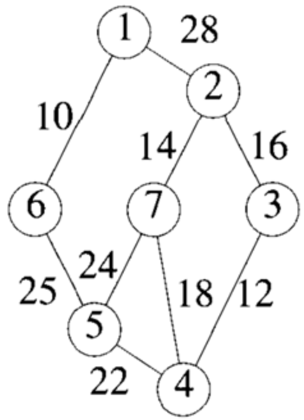
# Prim's Algorithm

- Initially add the least cost edge

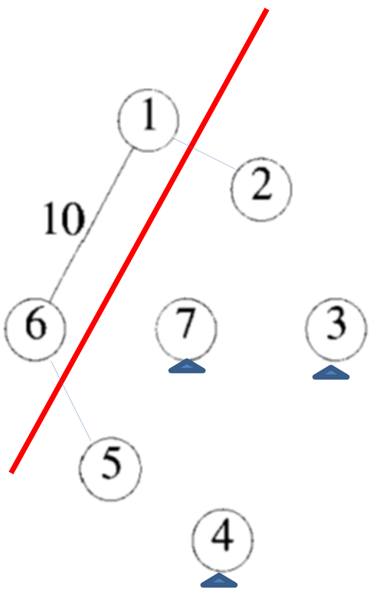


# Prim's Algorithm

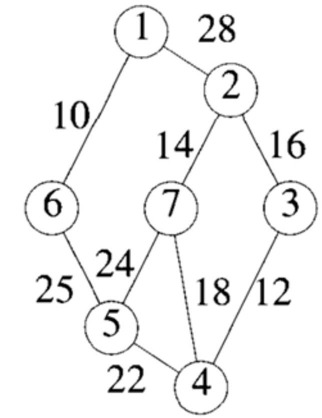



- $t = \{1,6\}$ is the tree vertex set.
- $v = \{2,3,4,5,7\}$ is the pending vertex set.
- Choose a least cost edge $(i, j)$
  - $i \in t, j \in v$.
  - the resulting spanning tree continues to be a tree (no cycles are formed by adding the new edge).
  - The cost of spanning tree continues to be minimum.

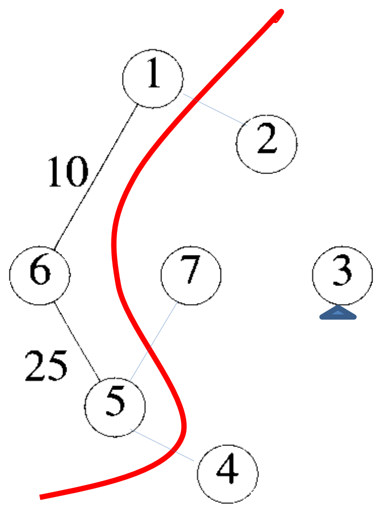



| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 6 | 0 | 1 |
|  | 28 | ∞ | ∞ | 25 |  | ∞ |

# Prim's Algorithm

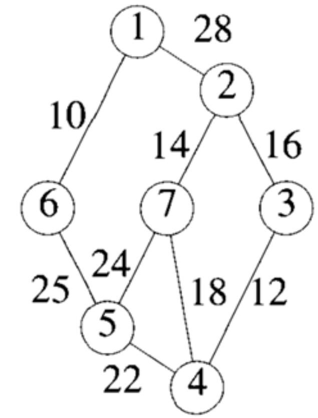



- $t = \{1,5,6\}$ is the tree vertex set.
- $v = \{2,3,4,7\}$ is the pending vertex set.
- Choose a least cost edge $(i, j)$
  - $i \in t, j \in v$.
  - the resulting spanning tree continues to be a tree (no cycles are formed by adding the new edge).
  - The cost of spanning tree continues to be minimum.

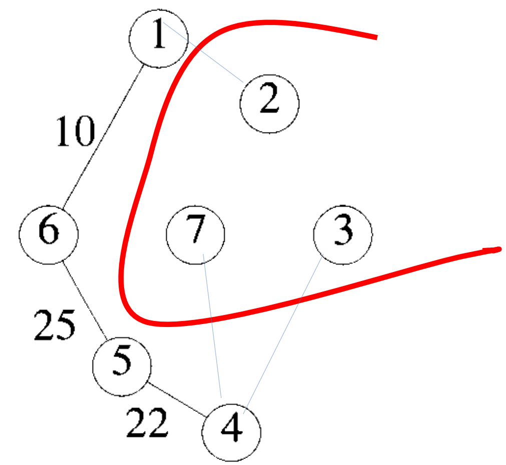



| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 5 | 0 | 0 | 5 |
| | 28 | $\infty$ | 22 | | | 24 |

# Prim's Algorithm

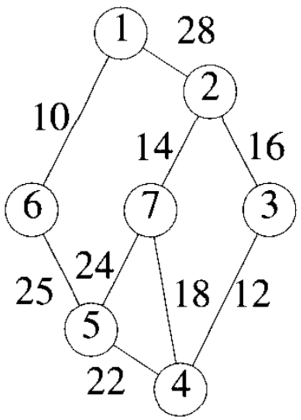


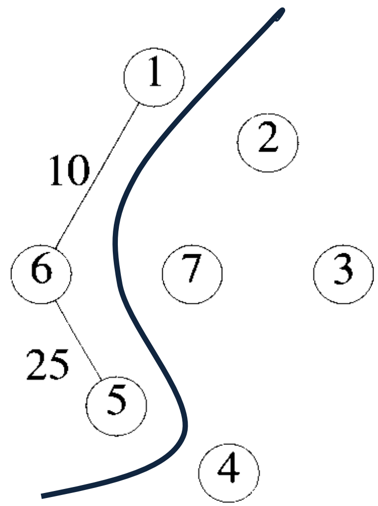

- $t = \{1,4,5,6\}$ is the tree vertex set.
- $v = \{2,3,7\}$ is the pending vertex set.
- Choose a least cost edge $(i, j)$
  - $i \in t, j \in v$.
  - the resulting spanning tree continues to be a tree (no cycles are formed by adding the new edge).
  - The cost of spanning tree continues to be minimum.




| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 1 | 4 | 0 | 0 | 0 | 4 |
| | 28 | 12 | | | | 18 |

# Prim's Algorithm




Pending vertex

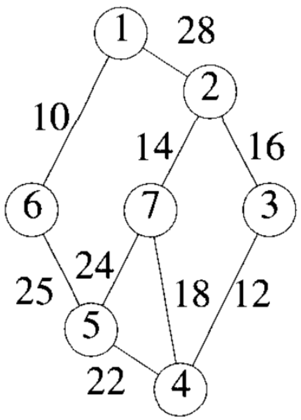
**near**

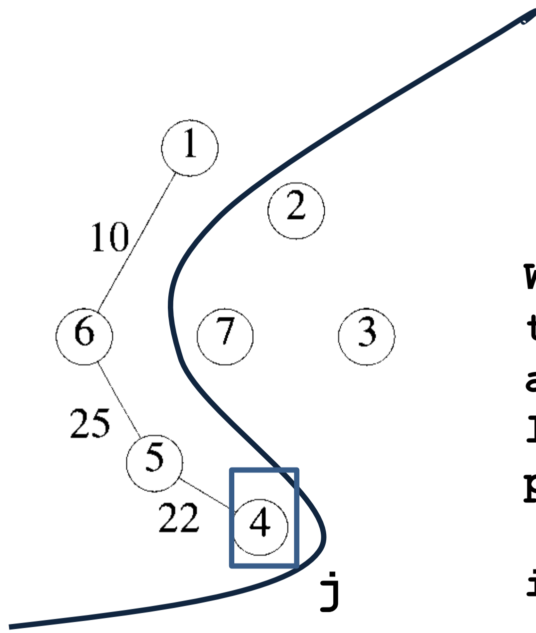
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 5 | 0 | 0 | 5 |

tree vertex

1. **near[i]:=0 if i is a tree vertex.**
2. **For each pending vertex i, near[i] is a tree vertex such that cost[i,near[i]] is the least cost edge connecting i to tree vertex near[i].**

# Prim's Algorithm




**near**

**i** →

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

**Whenever a new vertex joins the tree vertex set, update the near array so that it maintains the least cost edges connecting a pending vertex to a tree vertex.**

```
if (cost[i,j]<cost[i,near[i])then
       near[i]:=j;
```

# Prim's Algorithm

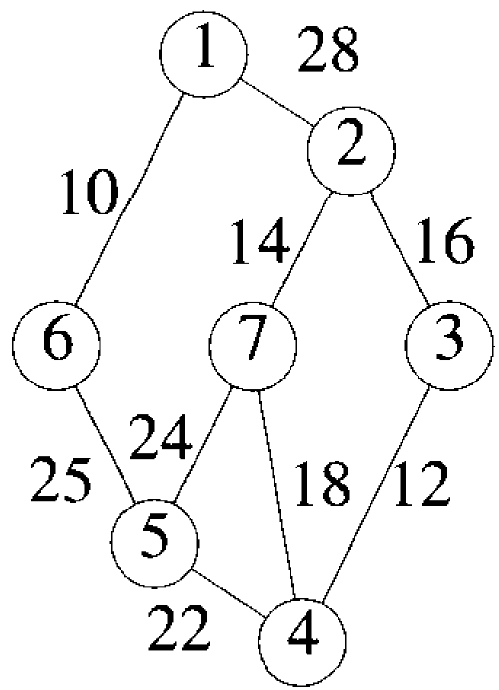
**Least cost edge**

```
Choose j such that near[j] ≠ 0 and cost[j,near[j]] is minimum
Store the edge (j,near[j])
near[j]:=0
for i:= 1 to n do
   if (near[i] ≠ 0 and cost[i,near[i]] >cost[i,j]) then
      near[i]:=j;
```

**Existing edge**

**Edge with new vertex**

# Prim's Algorithm



# Prim's Algorithm

**near**

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

**G**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | 28 | | | | 10 | |
| 2 | 28 | | 16 | | | | 14 |
| 3 | | 16 | | 12 | | | |
| 4 | | | 12 | | 22 | | 18 |
| 5 | | | | 22 | | 25 | 24 |
| 6 | 10 | | | | 25 | | |
| 7 | | 14 | | 18 | 24 | | |

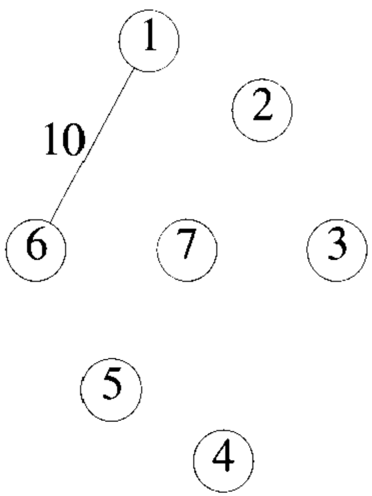
**Choose the least cost edge.**
**Add it to the spanning tree.**
**Initialize the near array**

# Prim's Algorithm

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| **near** | 0 | 1(28) | 1(∞) | 1(∞) | 6(25) | 0 | 1(∞) |

**G**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | 28 | | | | 10 | |
| 2 | 28 | | 16 | | | | 14 |
| 3 | | 16 | | 12 | | | |
| 4 | | | 12 | | 22 | | 18 |
| 5 | | | | 22 | | 25 | 24 |
| 6 | 10 | | | | 25 | | |
| 7 | | 14 | | 18 | 24 | | |




`mincost:=10;`

**t**

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

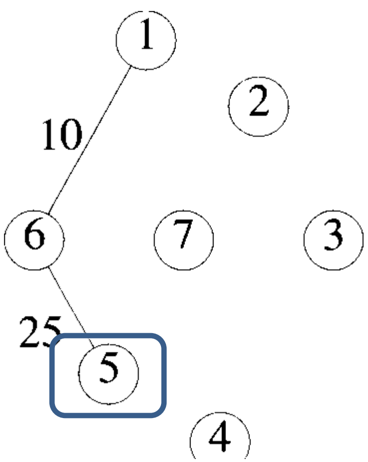
**Choose the nearest pending vertex.**
**Add to the spanning tree.**
**Update the near array.**

# Prim's Algorithm

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| **near** | 0 | 1(28) | 1(∞) | 1(∞) | 6(25) | 0 | 1(∞) |

**G**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | 28 | | | | 10 | |
| 2 | 28 | | 16 | | | | 14 |
| 3 | | 16 | | 12 | | | |
| 4 | | | 12 | | 22 | | 18 |
| 5 | | | | 22 | | 25 | 24 |
| 6 | 10 | | | | 25 | | |
| 7 | | 14 | | 18 | 24 | | |




mincost:=35;

t

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

Choose the nearest pending vertex.
Add to the spanning tree.
Update the near array.

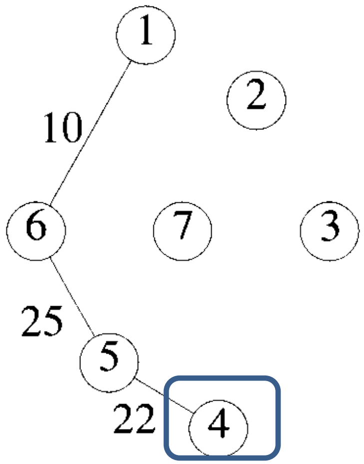
# Prim's Algorithm

**near**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| | 0 | 1(26) | 1(∞) | 1(∞) | 6(25) | 0 | 1(∞) |
| | 0 | 1(28) | 1(∞) | 5(22) | 0 | 0 | 5(24) |
| | | | | | | | |

**G**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | 28 | | | | 10 | |
| 2 | 28 | | 16 | | | | 14 |
| 3 | | 16 | | 12 | | | |
| 4 | | | 12 | | 22 | | 18 |
| 5 | | | | 22 | | 25 | 24 |
| 6 | 10 | | | | 25 | | |
| 7 | | 14 | | 18 | 24 | | |




**t**

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

`mincost:=57;`

```
Choose the nearest
pending vertex.
Add to the spanning
tree.
Update the near array.
```

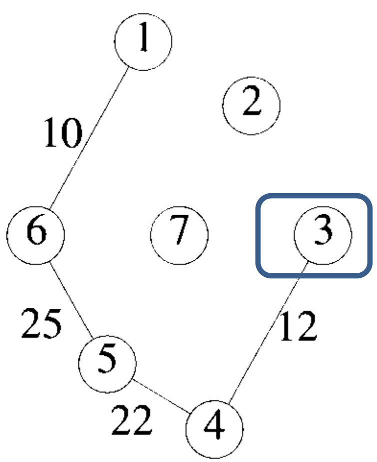
# Prim's Algorithm

near

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 1(28) | 1(∞) | 1(∞) | 6(25) | 0 | 1(∞) |
| 0 | 1(28) | 1(∞) | 5(22) | 0 | 0 | 5(24) |
| 0 | 1(28) | 4(12) | 0 | 0 | 0 | 4(18) |
| | | | | | | |

G

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | 28 | | | | 10 | |
| 2 | 28 | | 16 | | | | 14 |
| 3 | | 16 | | 12 | | | |
| 4 | | | 12 | | 22 | | 18 |
| 5 | | | | 22 | | 25 | 24 |
| 6 | 10 | | | | 25 | | |
| 7 | | 14 | | 18 | 24 | | |




t

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

mincost:=69;

Choose the nearest pending vertex.
Add to the spanning tree.
Update the near array.

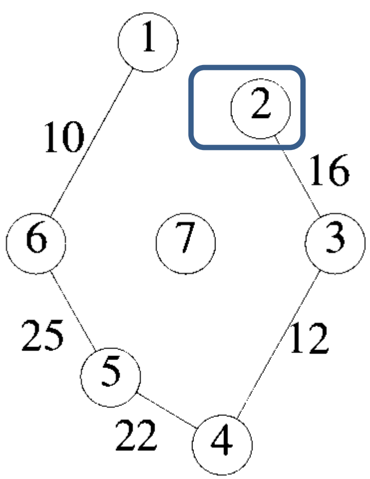
# Prim's Algorithm

**near**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 1(28) | 1($\infty$) | 1($\infty$) | 6(25) | 0 | 1($\infty$) |
| 0 | 1(28) | 1($\infty$) | 5(22) | 0 | 0 | 5(24) |
| 0 | 1(28) | 4(12) | 0 | 0 | 0 | 4(18) |
| 0 | 3(16) | 0 | 0 | 0 | 0 | 4(18) |
| | | | | | | |

**G**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | 28 | | | | 10 | |
| 2 | 28 | | 16 | | | | 14 |
| 3 | | 16 | | 12 | | | |
| 4 | | | 12 | | 22 | | 18 |
| 5 | | | | 22 | | 25 | 24 |
| 6 | 10 | | | | 25 | | |
| 7 | | 14 | | 18 | 24 | | |




**t**

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

Choose the nearest pending vertex.
Add to the spanning tree.
Update the near array.

`mincost:=85;`

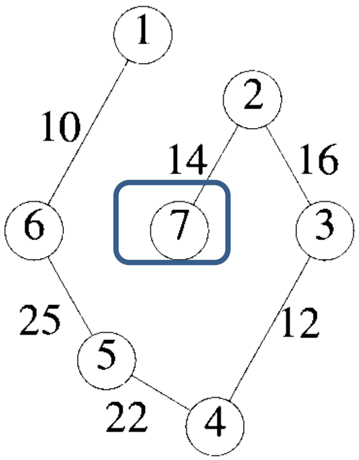
# Prim's Algorithm

**near**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 1(26) | 1(∞) | 1(∞) | 6(25) | 0 | 1(∞) |
| 0 | 1(26) | 1(∞) | 5(22) | 0 | 0 | 5(24) |
| 0 | 1(26) | 4(12) | 0 | 0 | 0 | 4(18) |
| 0 | 3(16) | 0 | 0 | 0 | 0 | 4(18) |
| 0 | 0 | 0 | 0 | 0 | 0 | 2(14) |

**G**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | 28 | | | | 10 | |
| 2 | 28 | | 16 | | | | 14 |
| 3 | | 16 | | 12 | | | |
| 4 | | | 12 | | 22 | | 18 |
| 5 | | | | 22 | | 25 | 24 |
| 6 | 10 | | | | 25 | | |
| 7 | | 14 | | 18 | 24 | | |

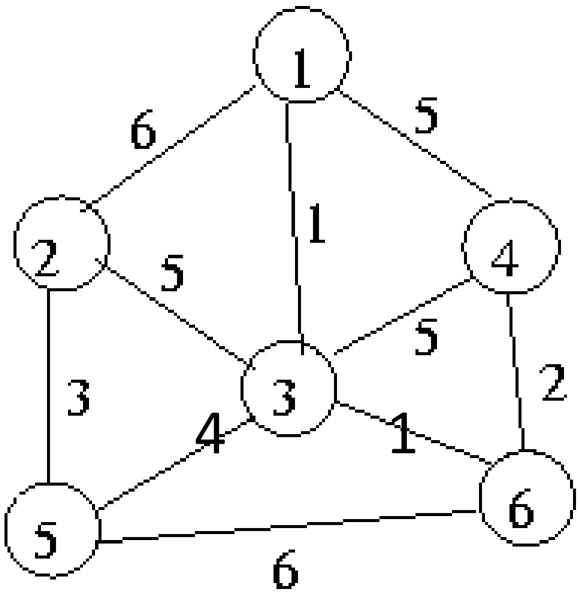



**t**

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

`mincost:=99;`

# Prim's algorithm




$$
\begin{bmatrix}
\infty & 6 & 1 & 5 & \infty & \infty \\
6 & \infty & 5 & \infty & 3 & \infty \\
1 & 5 & \infty & 5 & 4 & 1 \\
5 & \infty & 5 & \infty & \infty & 2 \\
\infty & 3 & 4 & \infty & \infty & 6 \\
\infty & \infty & 1 & 2 & 6 & \infty
\end{bmatrix}
$$

# Prim's algorithm

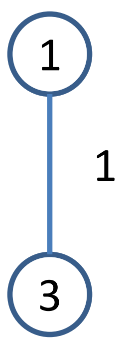
$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 4 & 1 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 4 & \infty & \infty & 6 \\ \infty & \infty & 1 & 2 & 6 & \infty \end{bmatrix}$$

**near**

| 1 | 2 | 3 | 4 | 5 | 6 | j |
|---|---|---|---|---|---|---|
| 0 | 3(5) | 0 | 1(5) | 3(4) | 3(1) | |

**t**

| | |
|---|---|
| 1 | 3 |
| | |
| | |
| | |
| | |




**Choose j such that near[j]≠ 0 and cost[j,near[j]] is minimum**
**near[j]:=0**

# Prim's algorithm

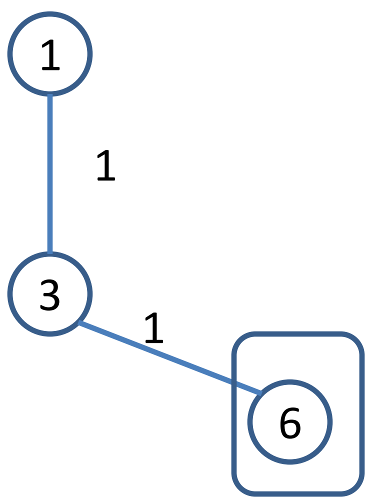
$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 4 & 1 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 4 & \infty & \infty & 6 \\ \infty & \infty & 1 & 2 & 6 & \infty \end{bmatrix}$$

**near**

| 1 | 2 | 3 | 4 | 5 | 6 j |
|---|---|---|---|---|---|
| 0 | 3(5) | 0 | 1(5) | 3(4) | ~~3(1)~~ |
| | | | | | |

**t**

| | |
|---|---|
| 1 | 3 |
| 6 | 3 |
| | |
| | |
| | |




```
for i:= 1 to n do
    if (near[i]≠0 and cost[i,near[i]] >
                        cost[i,j]) then
        near[i]:=j;
```

# Prim's algorithm

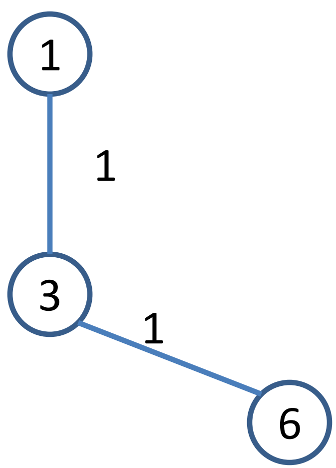
$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 4 & 1 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 4 & \infty & \infty & 6 \\ \infty & \infty & 1 & 2 & 6 & \infty \end{bmatrix}$$

**near**

| 1 | 2 | 3 | 4 j | 5 | 6 |
|---|---|---|---|---|---|
| 0 | 3(5) | 0 | 1(5) | 3(4) | 3(1) |
| 0 | 3(5) | 0 | 6(2) | 3(4) | 0 |

**t**

| | |
|---|---|
| 1 | 3 |
| 6 | 3 |
| | |
| | |
| | |




**Choose j such that near[j]≠ 0 and cost[j,near[j]] is minimum**
**near[j]:=0**

# Prim's algorithm

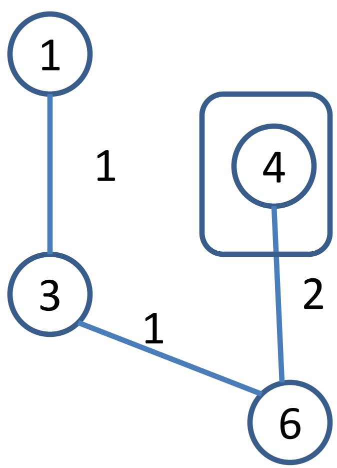
$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 4 & 1 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 4 & \infty & \infty & 6 \\ \infty & \infty & 1 & 2 & 6 & \infty \end{bmatrix}$$

**near**

| 1 | 2 | 3 | 4 j | 5 | 6 |
|---|---|---|---|---|---|
| 0 | 3(5) | 0 | 1(5) | 3(4) | 3(1) |
| 0 | 3(5) | 0 | ~~6(2)~~ | 3(4) | 0 |
| | | | | | |

**t**

| | |
|---|---|
| 1 | 3 |
| 6 | 3 |
| 4 | 6 |
| | |
| | |




```
for i:= 1 to n do
    if (near[i]≠0 and cost[i,near[i]] >
                          cost[i,j]) then
        near[i]:=j;
```

# Prim's algorithm

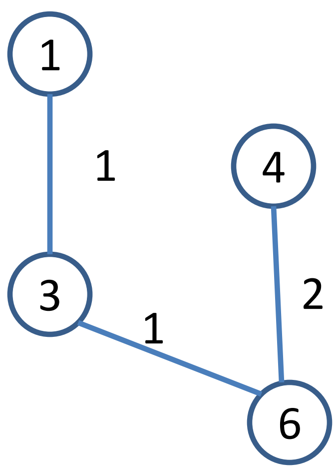
$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 4 & 1 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 4 & \infty & \infty & 6 \\ \infty & \infty & 1 & 2 & 6 & \infty \end{bmatrix}$$

**near**

| 1 | 2 | 3 | 4 | 5 **j** | 6 |
|---|---|---|---|---|---|
| 0 | 3(5) | 0 | 1(5) | 3(4) | 3(1) |
| 0 | 3(5) | 0 | 6(2) | 3(4) | 0 |
| 0 | 3(5) | 0 | 0 | 3(4) | 0 |

**t**

| | |
|---|---|
| 1 | 3 |
| 6 | 3 |
| 4 | 6 |
| | |
| | |




**Choose j such that near[j]≠ 0 and cost[j,near[j]] is minimum**
**near[j]:=0**

# Prim's algorithm

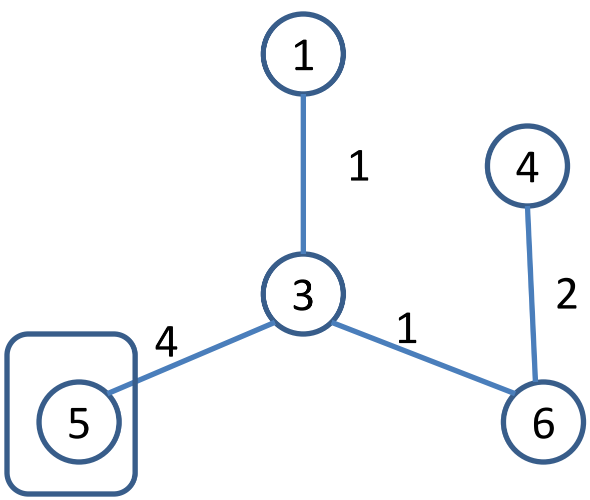
$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 4 & 1 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 4 & \infty & \infty & 6 \\ \infty & \infty & 1 & 2 & 6 & \infty \end{bmatrix}$$

**near**

| 1 | 2 | 3 | 4 | 5 j | 6 |
|---|---|---|---|---|---|
| 0 | 3(5) | 0 | 1(5) | 3(4) | 3(1) |
| 0 | 3(5) | 0 | 6(2) | 3(4) | 0 |
| 0 | 3(5) | 0 | 0 | ~~3(4)~~ | 0 |
| | | | | | |

**t**

| | |
|---|---|
| 1 | 3 |
| 6 | 3 |
| 4 | 6 |
| 5 | 3 |
| | |




```
for i:= 1 to n do
    if (near[i]≠ 0 and cost[i,near[i]] >
                         cost[i,j]) then
        near[i]:=j;
```

# Prim's algorithm

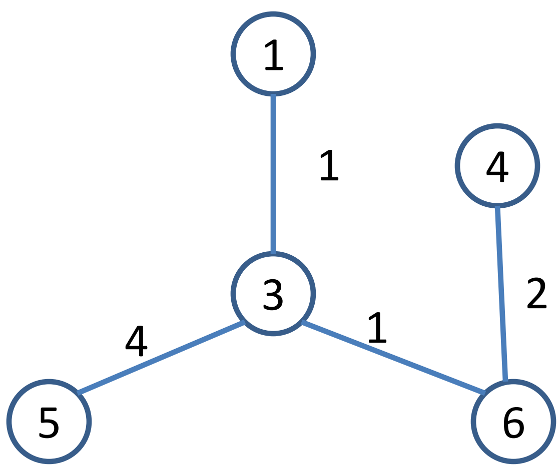
$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 4 & 1 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 4 & \infty & \infty & 6 \\ \infty & \infty & 1 & 2 & 6 & \infty \end{bmatrix}$$

**near**

| 1 | 2 j | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 0 | 3(5) | 0 | 1(5) | 3(4) | 3(1) |
| 0 | 3(5) | 0 | 6(2) | 3(4) | 0 |
| 0 | 3(5) | 0 | 0 | 3(4) | 0 |
| 0 | 5(3) | 0 | 0 | 0 | 0 |

**t**

| | |
|---|---|
| 1 | 3 |
| 6 | 3 |
| 4 | 6 |
| 5 | 3 |
| | |




**Choose j such that near[j]≠ 0 and cost[j,near[j]] is minimum**
**near[j]:=0**

# Prim's algorithm

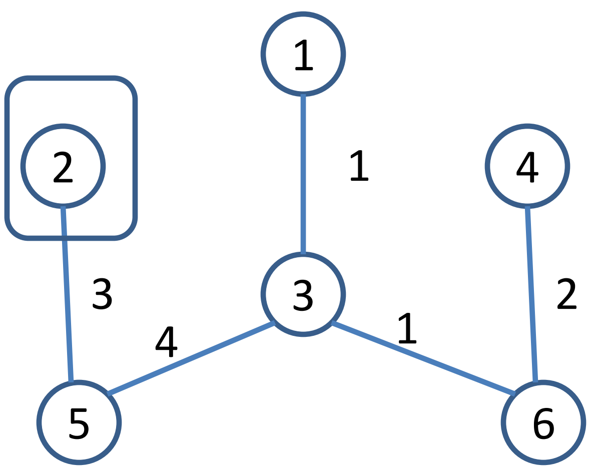
$$\begin{bmatrix} \infty & 6 & 1 & 5 & \infty & \infty \\ 6 & \infty & 5 & \infty & 3 & \infty \\ 1 & 5 & \infty & 5 & 4 & 1 \\ 5 & \infty & 5 & \infty & \infty & 2 \\ \infty & 3 & 4 & \infty & \infty & 6 \\ \infty & \infty & 1 & 2 & 6 & \infty \end{bmatrix}$$

**near**

| 1 | 2 j | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 0 | 3(5) | 0 | 1(5) | 3(4) | 3(1) |
| 0 | 3(5) | 0 | 6(2) | 3(4) | 0 |
| 0 | 3(5) | 0 | 0 | 3(4) | 0 |
| 0 | 5~~(3)~~ | 0 | 0 | 0 | 0 |
| | | | | | |

**t**

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




```
for i:= 1 to n do
    if (near[i] ≠ 0 and cost[i,near[i]] >
                              cost[i,j]) then
        near[i]:=j;
```

```
Algorithm Prim(E,cost,n,t)
//E is the set of edges. N is the number of vertices
//cost is 2-d array of size nxn such that cost[i,j] is the cost
//of the edge between the vertices i and j if exists else ∞
//t is a 2-d array of size (n-1)x2. if (p,q) is the ith edge
//included in the minimum cost spanning tree, then t[i,1=:=p
//and t[i,2]:=q;
```

```
{
    Let (u,v) be the minimum cost edge in E;
    mincost:=cost[u,v];
    t[1,1]:=u;t[1,2]:=v;
    for i:=1 to n do
        if(cost[i,u]<cost[i,v]) then  near[i]:=u;
        else near[i]:=v;
    near[u]:=0;near[v]:=0;
    for i:= 2 to n do
    {
        choose j such that near[j]≠ 0 and cost[j,near[j]) is
        the minimum;
        near[j]=0;
        mincost:=mincost+cost[j,near[j];
        t[i,1]:=j;t[i,2]:=near[j];
        for k:=1 to n do
            if(near[k] ≠ 0 and cost[k,j]<cost[k,near[k]) then
                near[k]:=j;
    }
    return mincost;
}
```

```
{
    Let (k,l) be the minimum cost edge in E;  O(|E|)
    mincost:=cost[k,l];
    t[1,1]:=k;t[1,2]:=l;
    for i:=1 to n do    O(n)
        if(cost[i,k]<cost[i,l]) then  near[i]:=k;
        else near[i]:=l;
    near[k]:=0;near[l]:=0;
    for i:= 2 to n-1 doO(n)
    {
        choose j such that near[j]≠0 and cost[j,near[j]) is
        minimum;
        near[j]=0;
        mincost:=mincost+cost[j,near[j];
        t[i,1]:=j;t[i,2]:=near[j];
        for k:=1 to n do  O(n)
            if(near[k] ≠ 0 and cost[k,j]<cost[k,near[k]) then
                near[k]:=j;
    }
}
return mincost;
```

$$O(n^2)$$