Minimal Spanning Tree (1/11)

- JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

Minimal Spanning Tree (1/11)

- JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

Six cities

Minimal Spanning Tree (1/11)

- JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

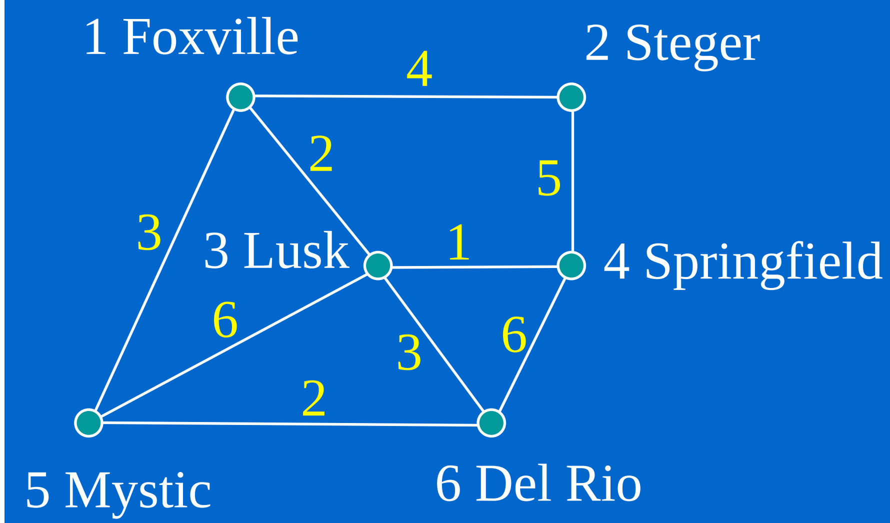
Six cities

We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

Six cities



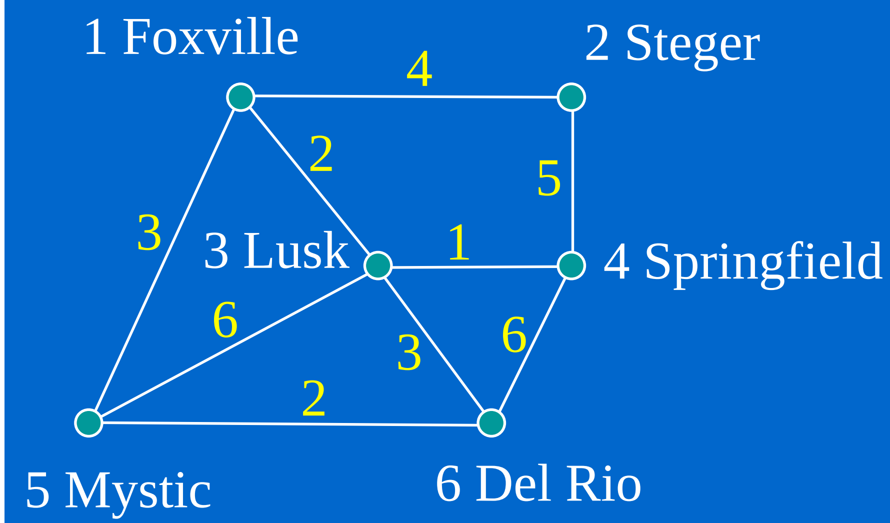
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

Minimal Spanning Tree (1/11)

- JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

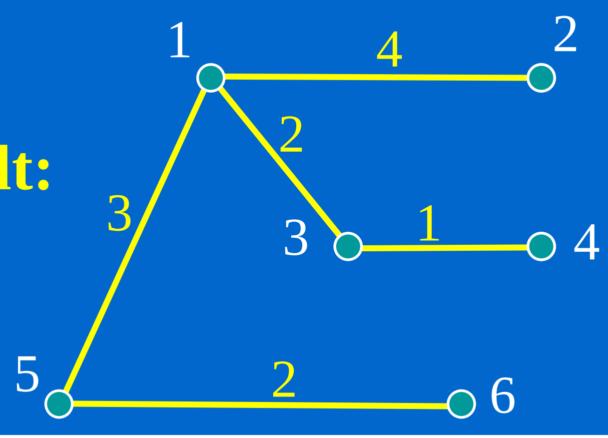
Six cities



We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

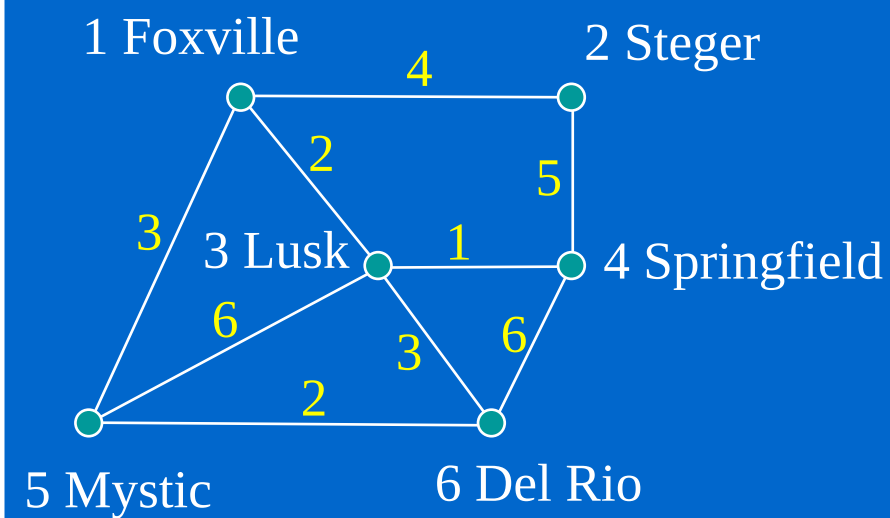
Result:



Minimal Spanning Tree (1/11)

- JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

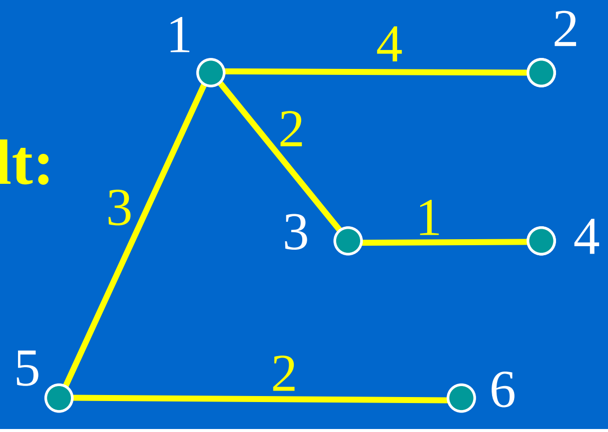
Six cities



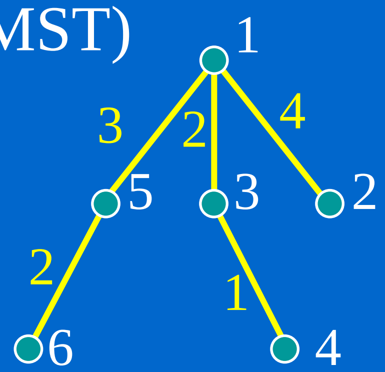
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

Result:



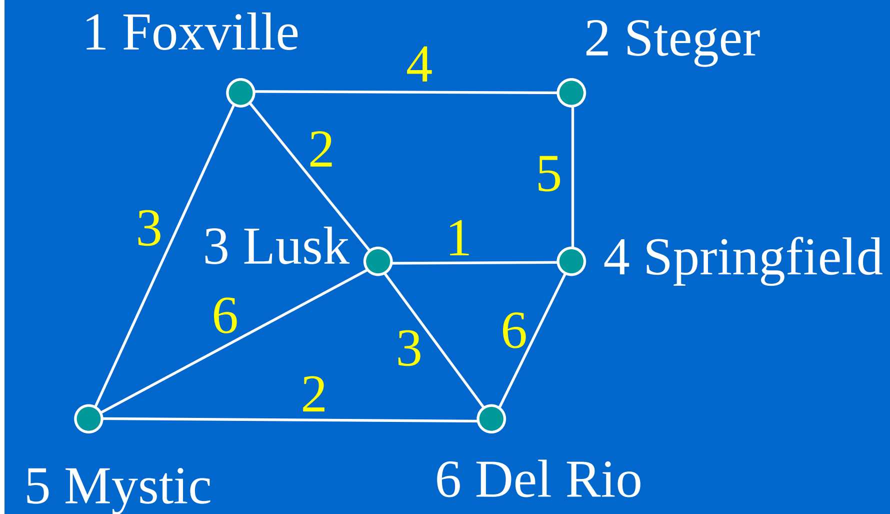
A tree (MST)



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

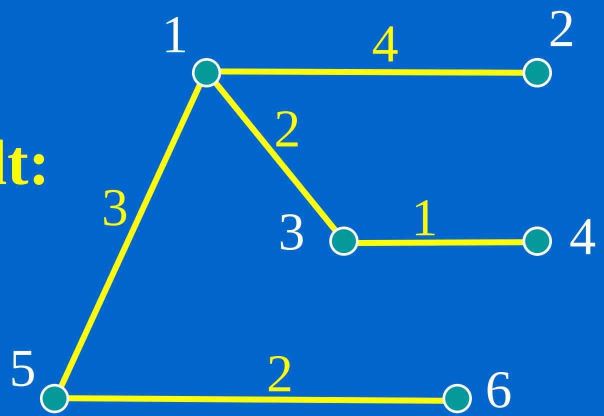
Six cities



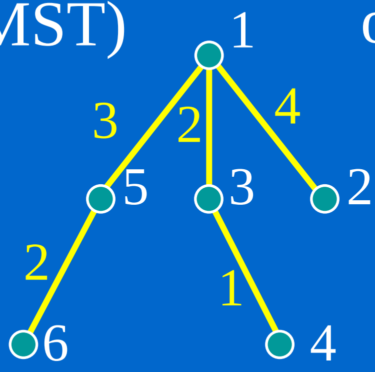
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

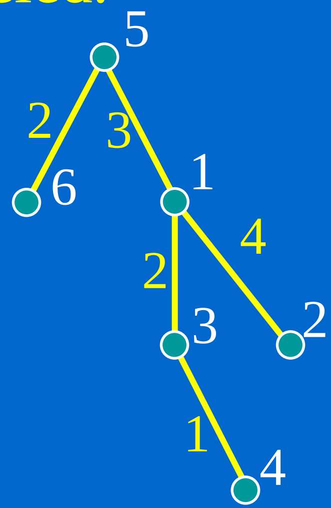
Result:



A tree (MST)



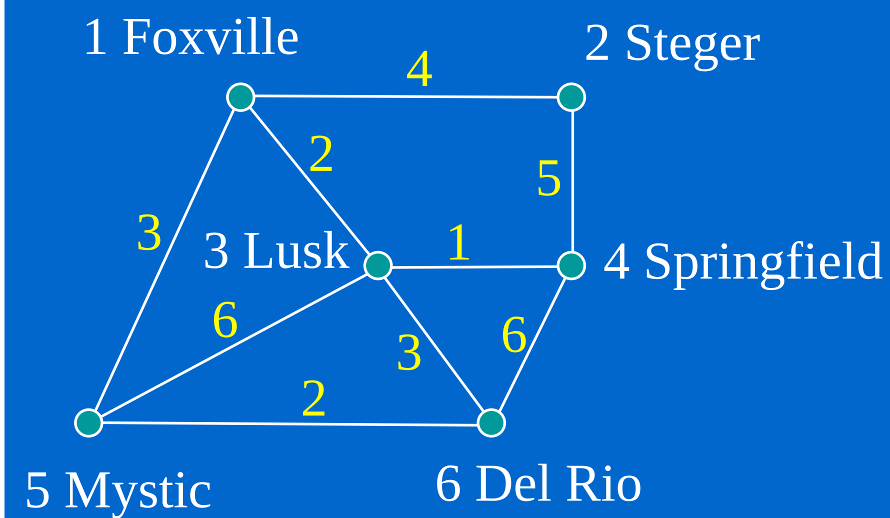
or



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

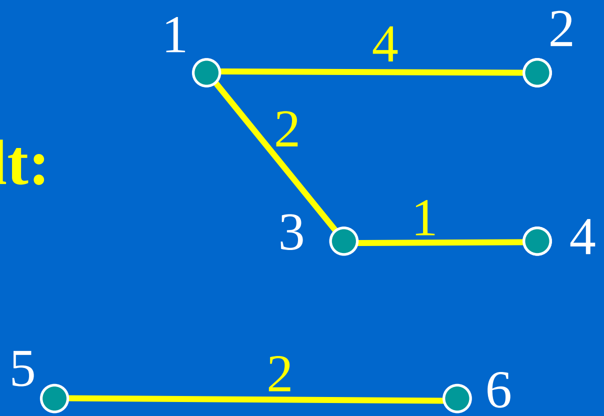
Six cities



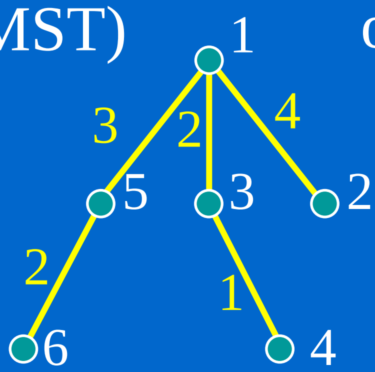
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

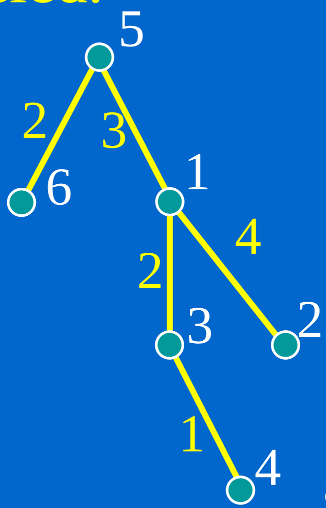
Result:



A tree (MST)



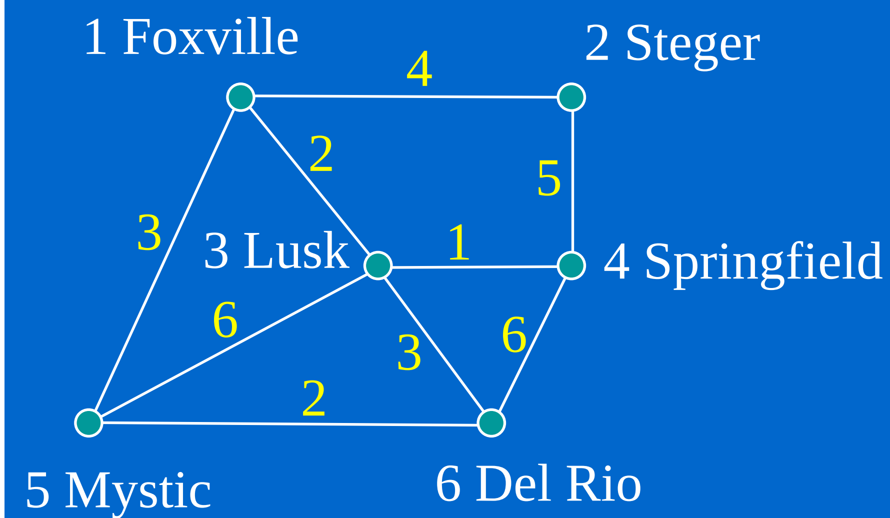
or



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

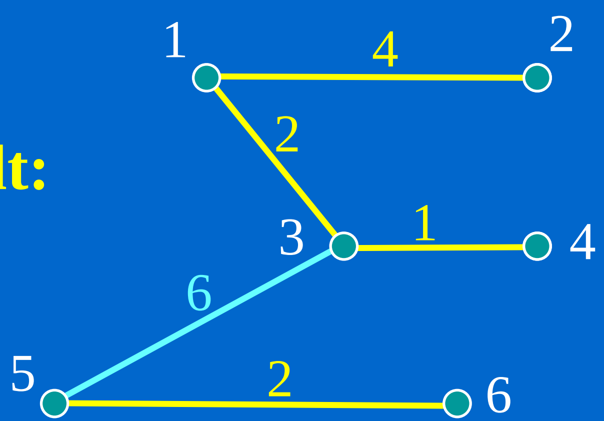
Six cities



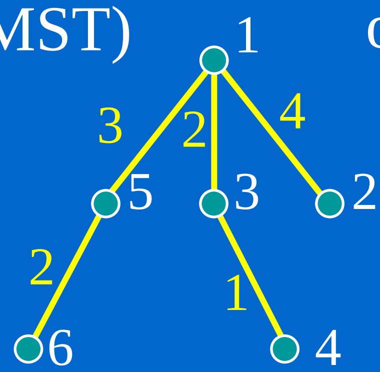
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

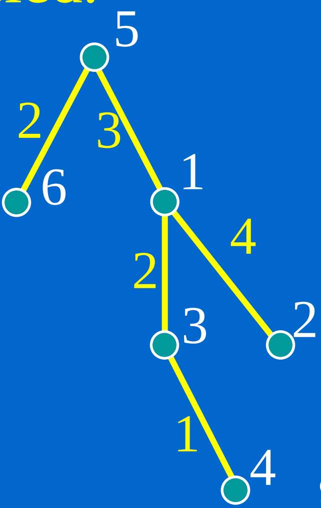
Result:



A tree (MST)



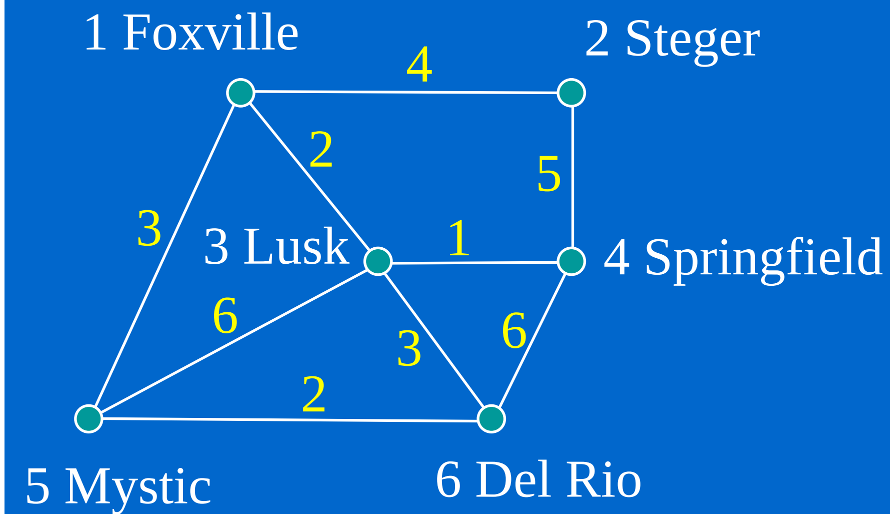
or



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

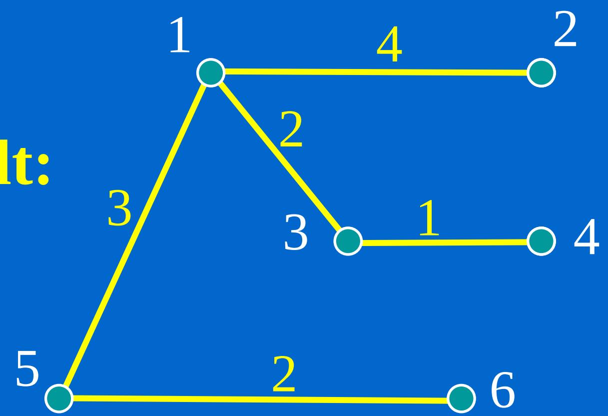
Six cities



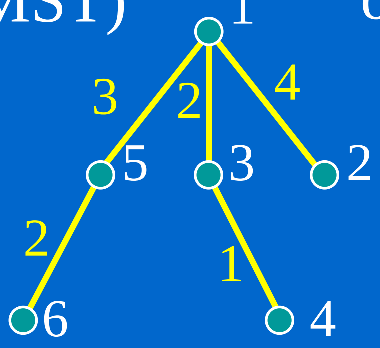
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

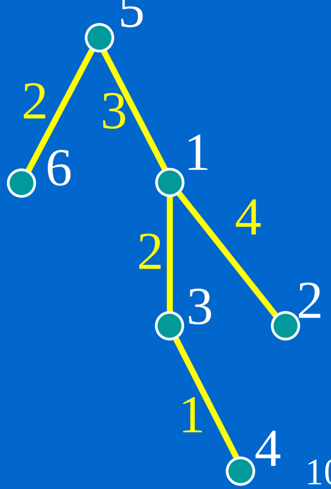
Result:



A tree (MST)



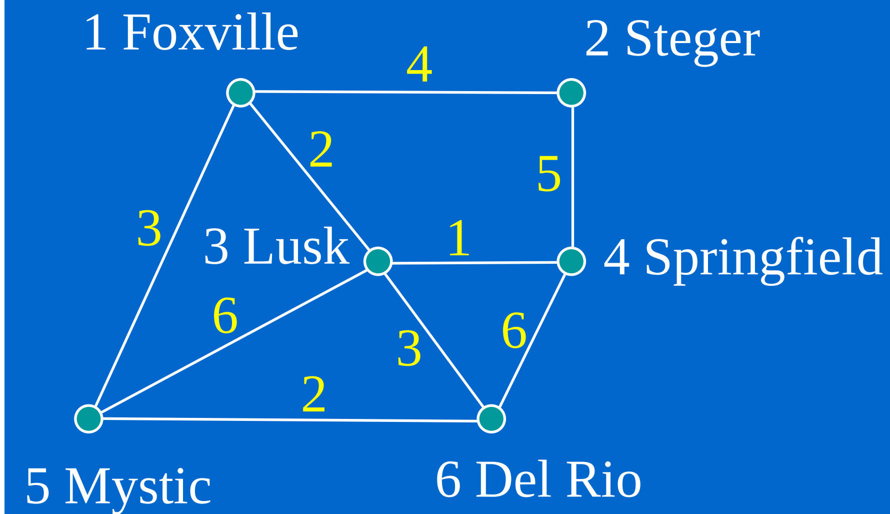
or



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

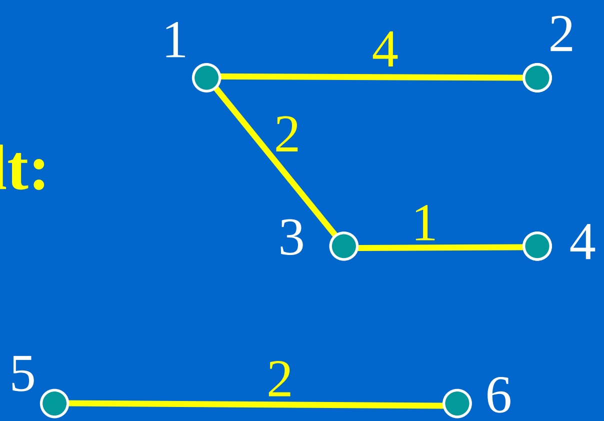
Six cities



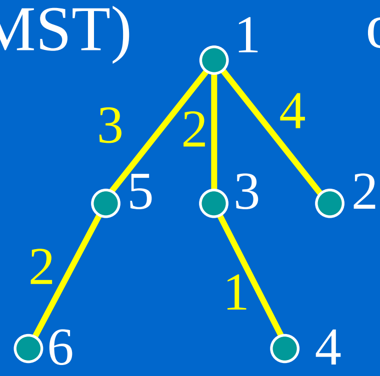
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

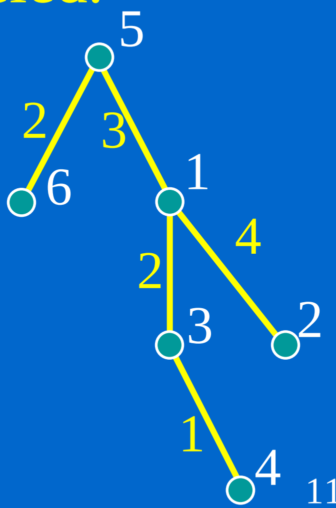
Result:



A tree (MST)



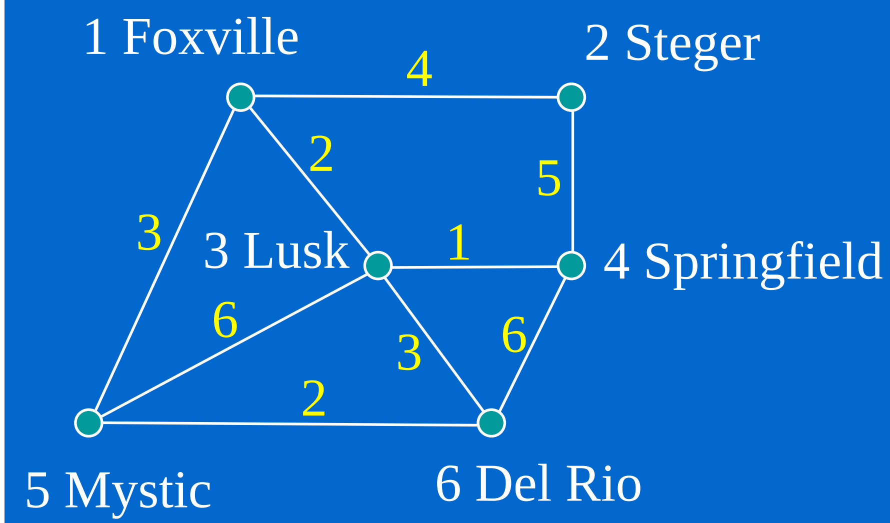
or



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

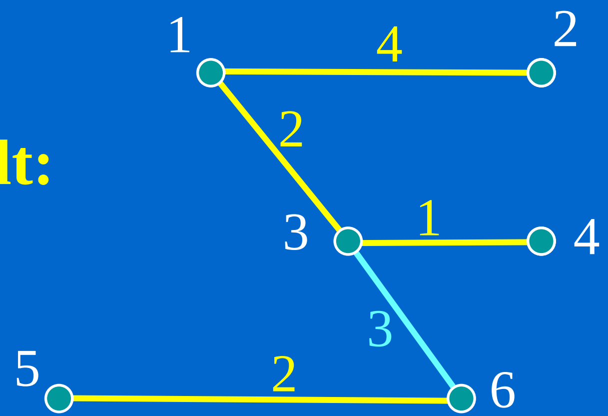
Six cities



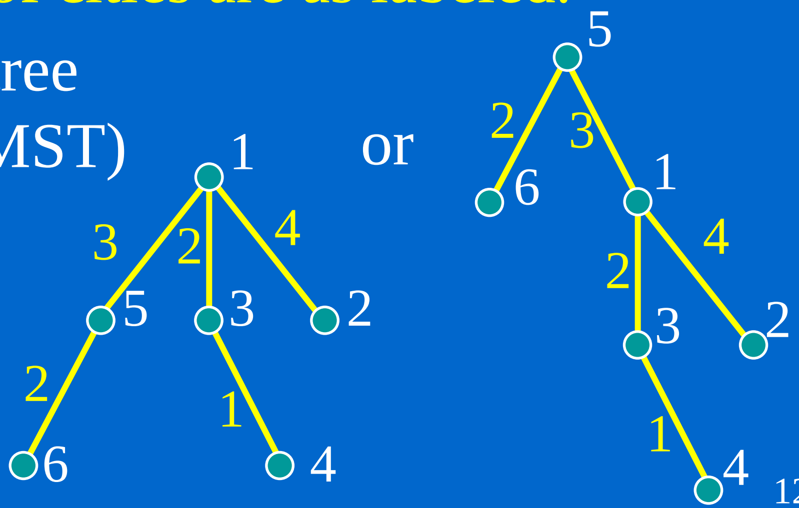
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

Result:



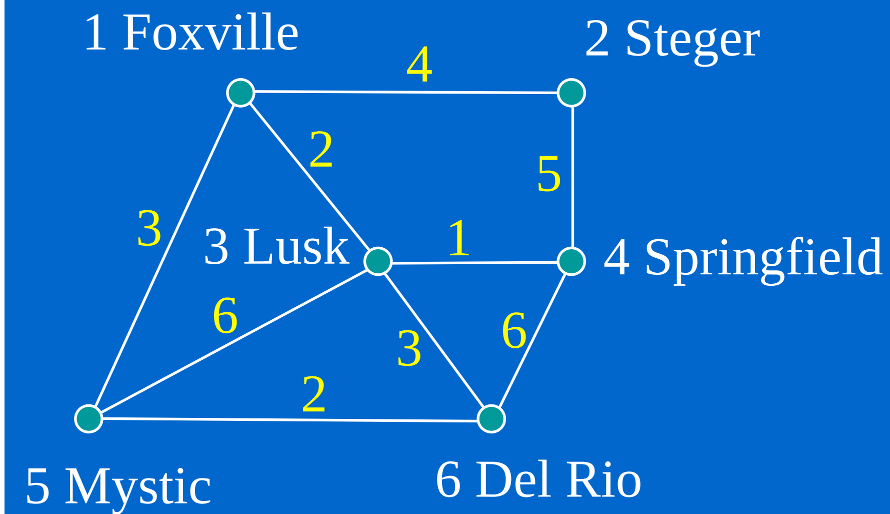
A tree (MST)



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

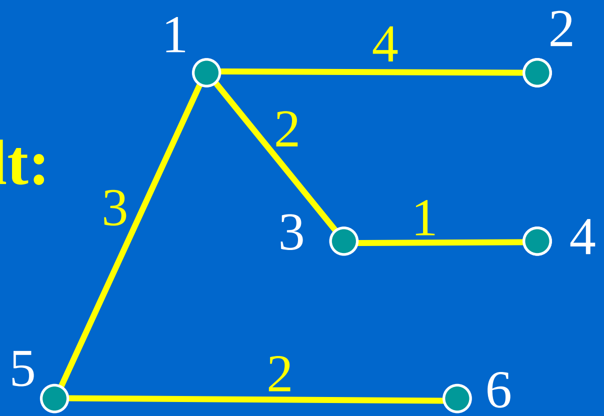
Six cities



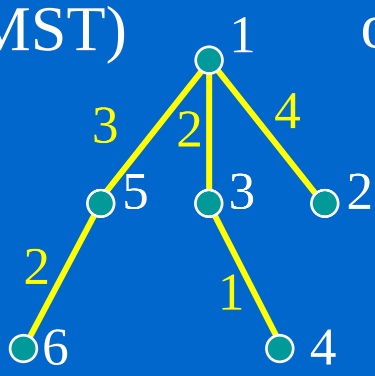
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

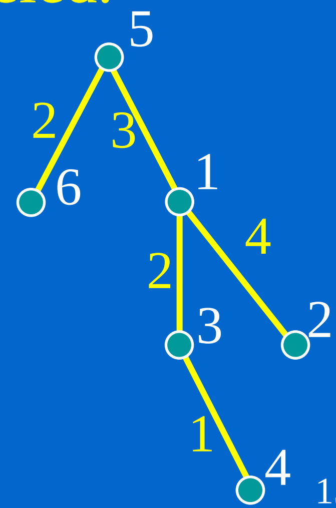
Result:



A tree (MST)



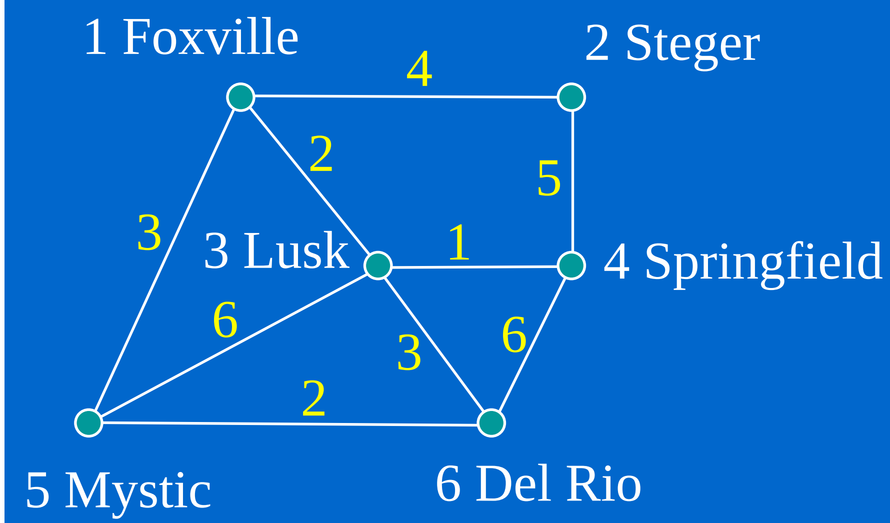
or



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

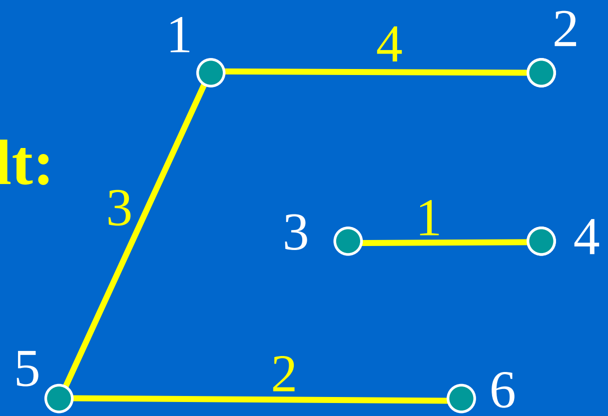
Six cities



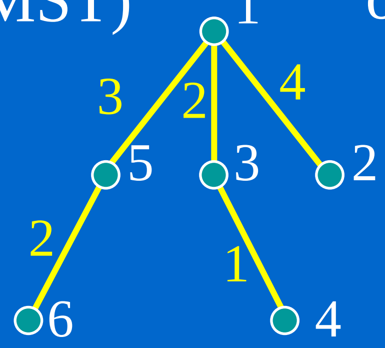
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

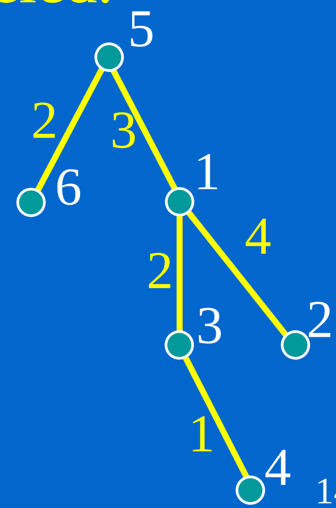
Result:



A tree (MST)



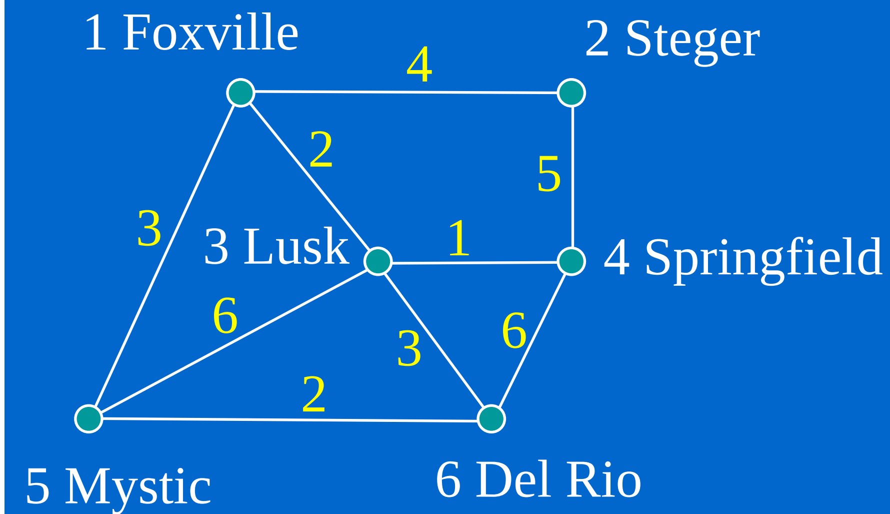
or



Minimal Spanning Tree (1/11)

- JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

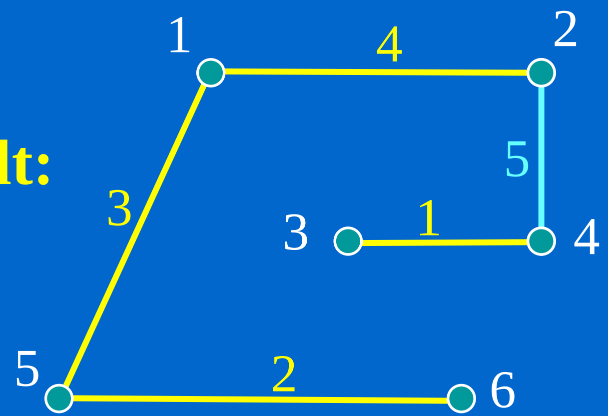
Six cities



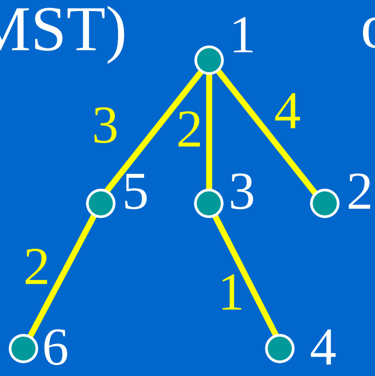
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

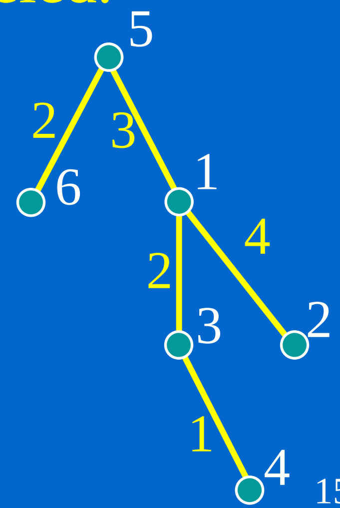
Result:



A tree (MST)



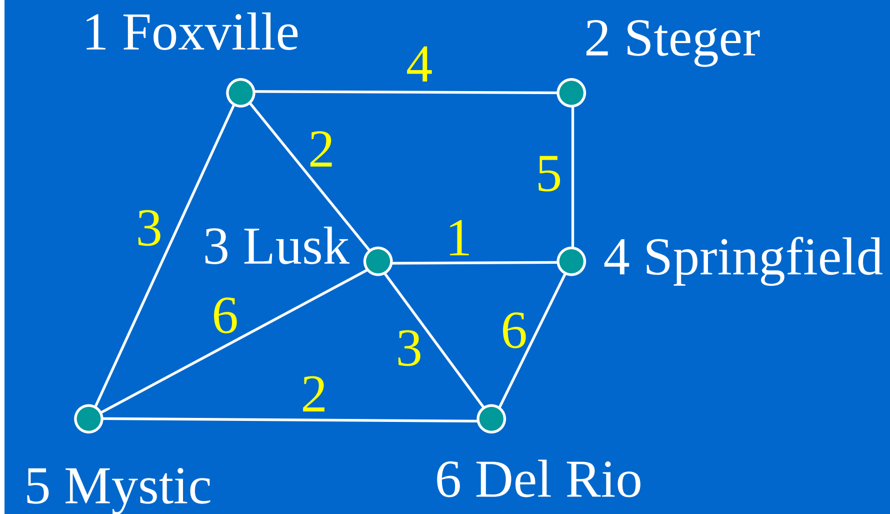
or



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

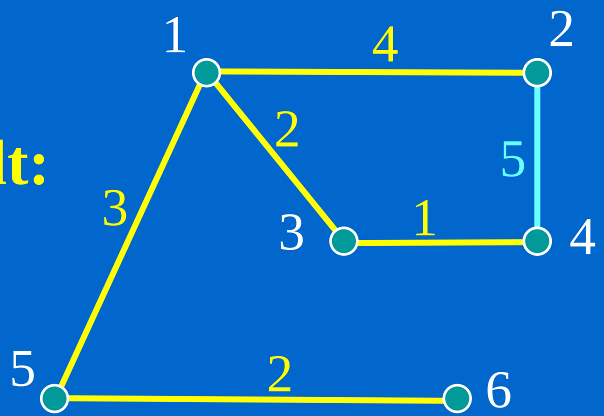
Six cities



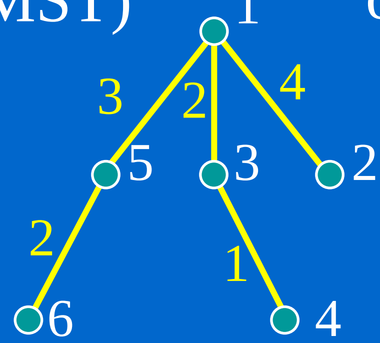
We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

The estimated costs for some pairs of cities are as labeled.

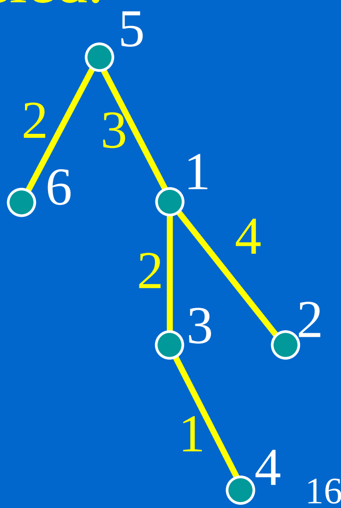
Result:



A tree (MST)



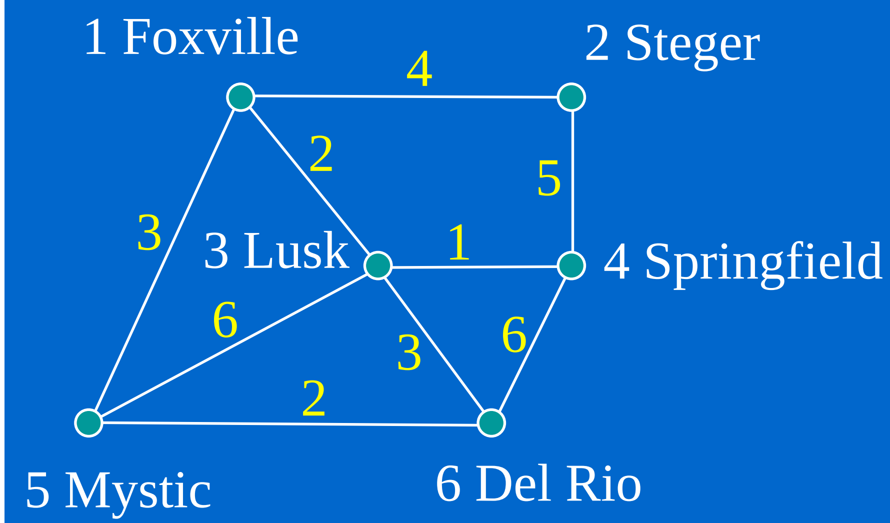
or



Minimal Spanning Tree (1/11)

✧ JohnsonBaugh's *Algorithms*, Section 7.2 (page 275) find Minimal Spanning Tree (MST) with **Kruskal's algorithm**:

Six cities

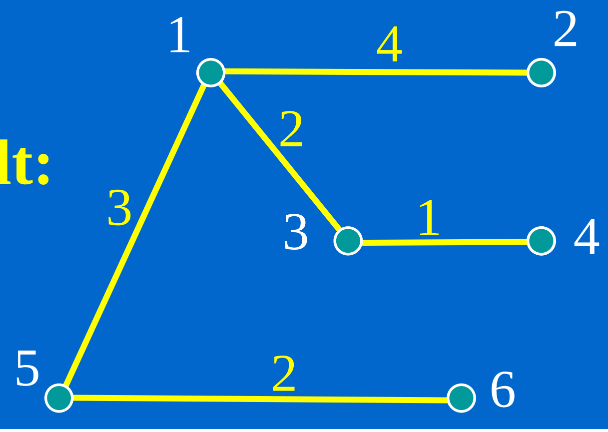


We want to construct a set of interconnecting roads such that one can reach any city from any starting city and the **total construction costs are minimized**.

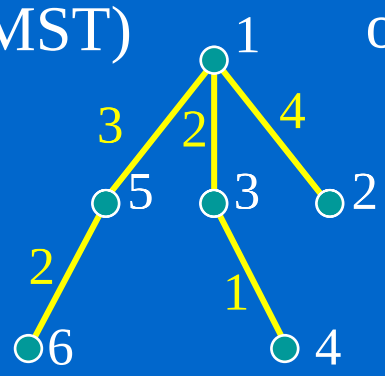
The estimated costs for some pairs of cities are as labeled.

Result:

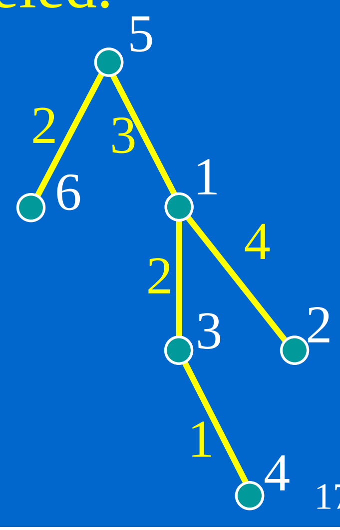
Best



A tree (MST)

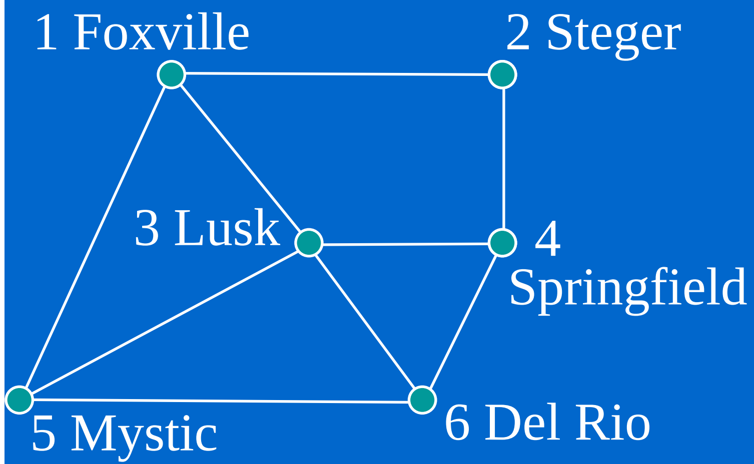


or



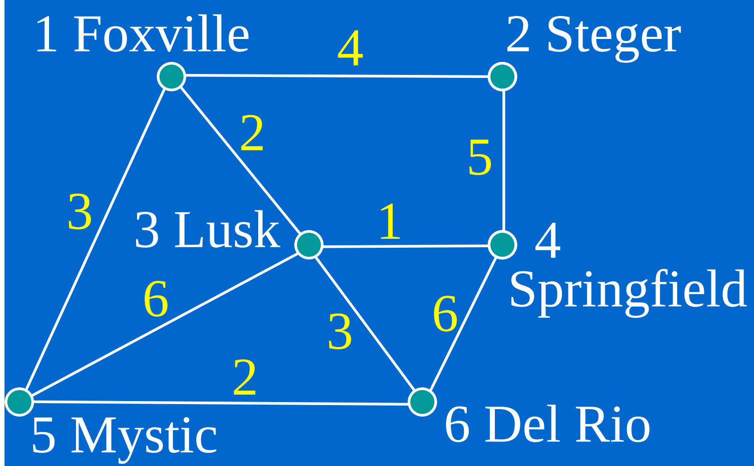
Kruskal's MST (2/11)

- **Kruskal's algorithm**



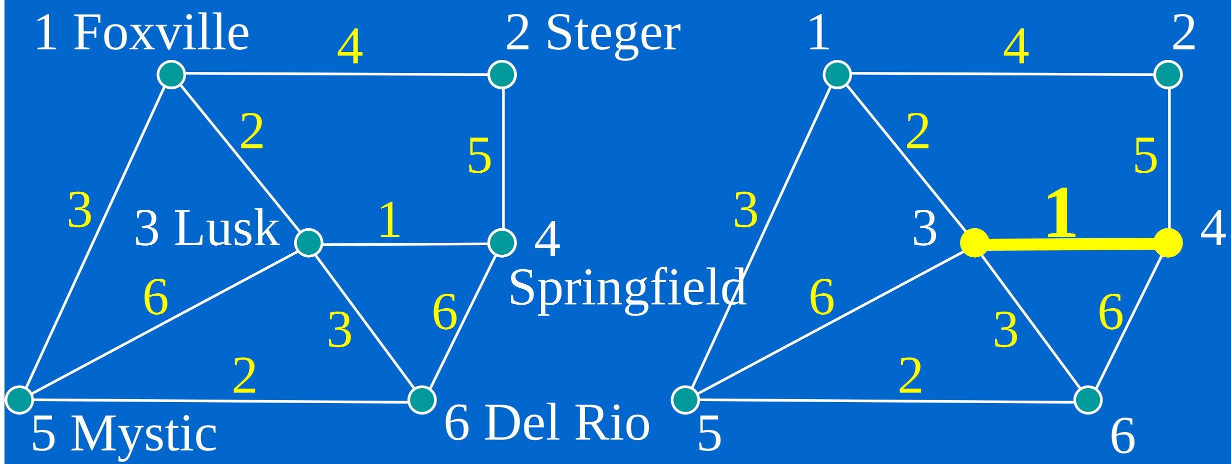
Kruskal's MST (2/11)

✧ **Kruskal's algorithm**



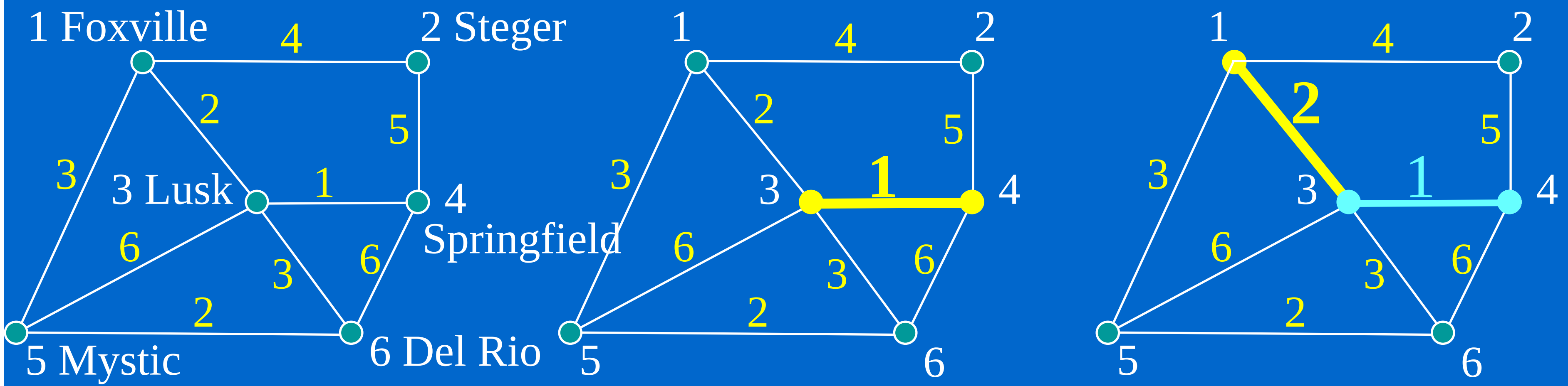
Kruskal's MST (2/11)

✧ Kruskal's algorithm



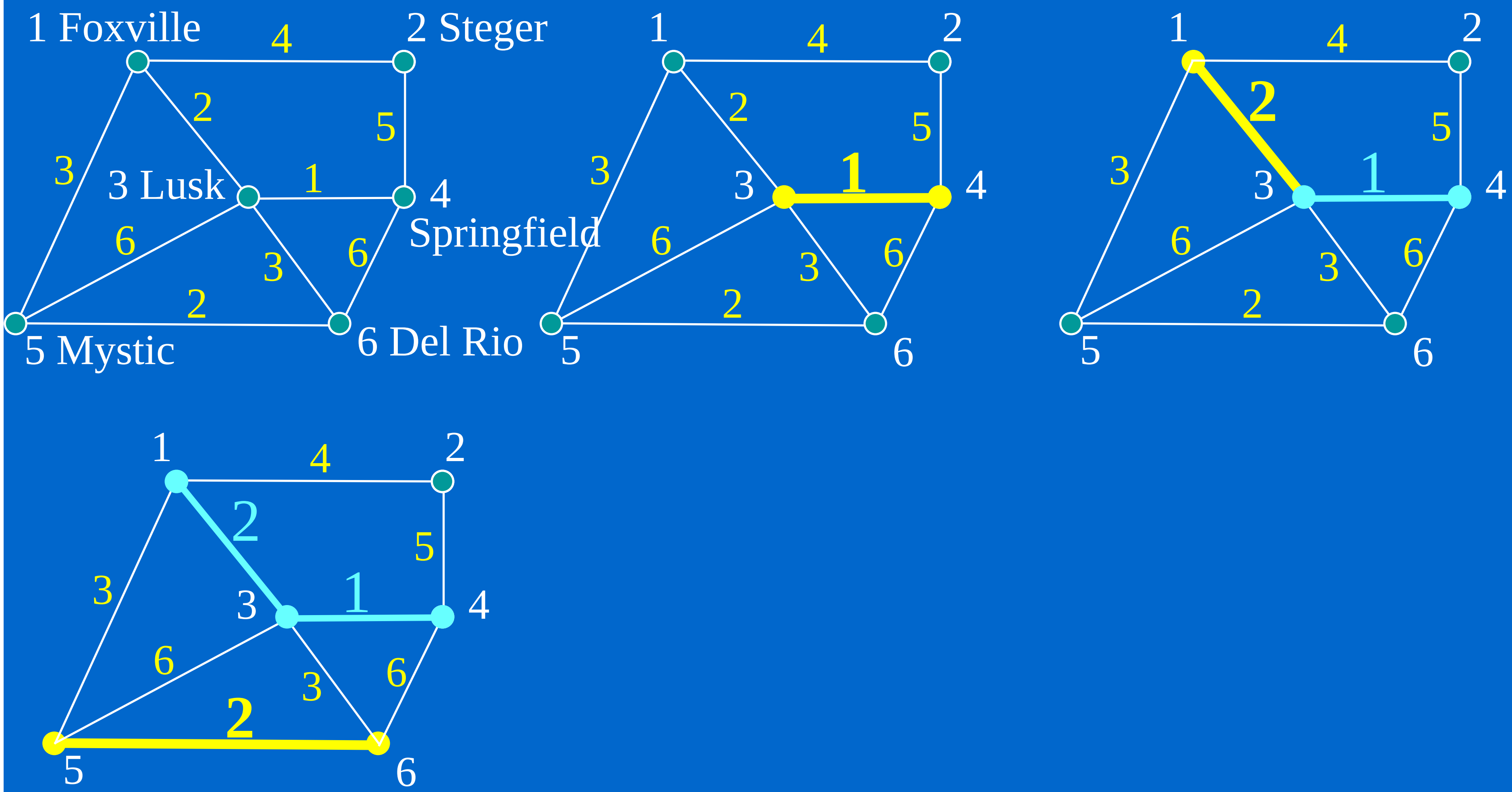
Kruskal's MST (2/11)

✧ Kruskal's algorithm



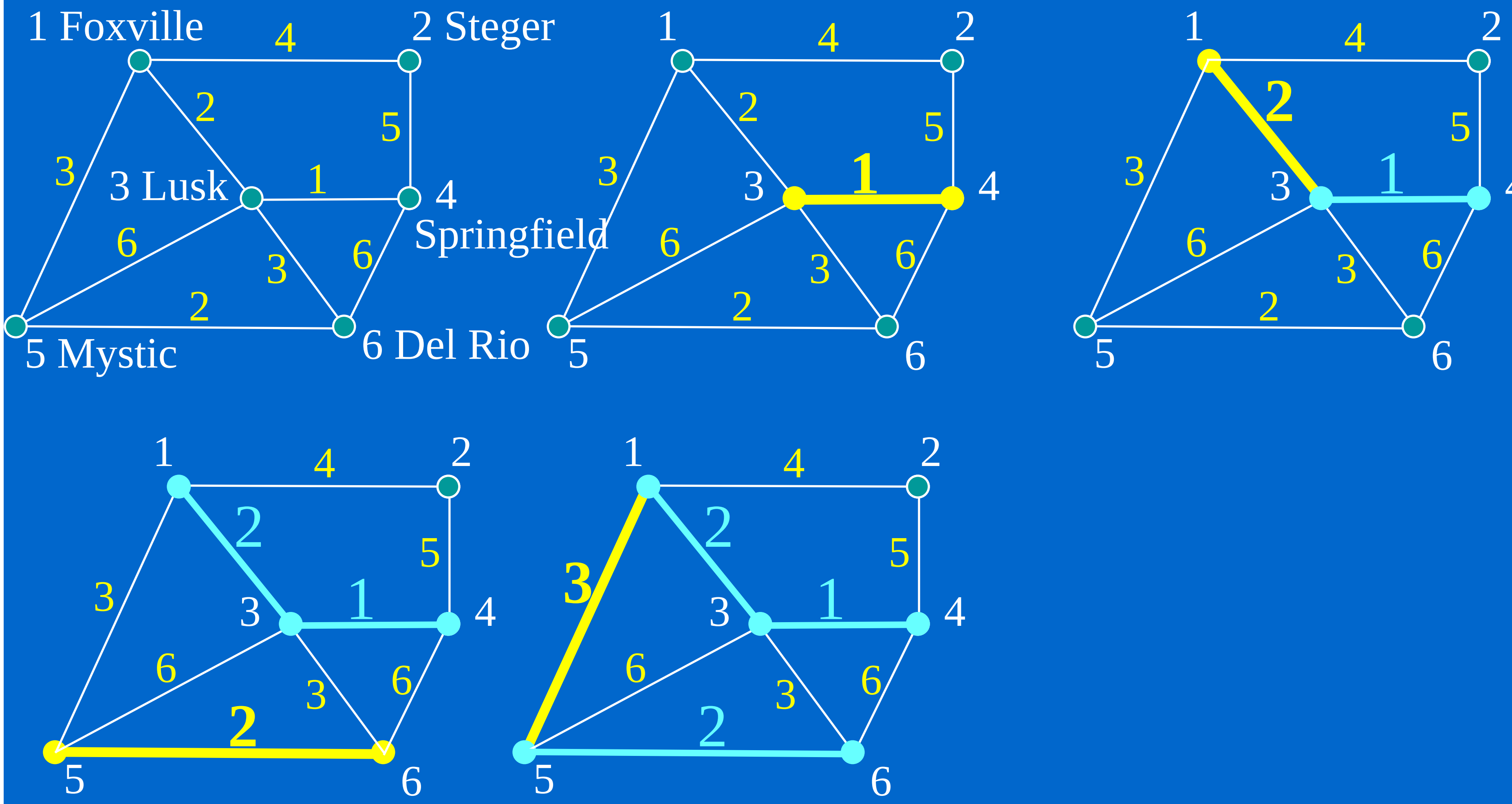
Kruskal’s MST (2/11)

✧ **Kruskal’s algorithm**



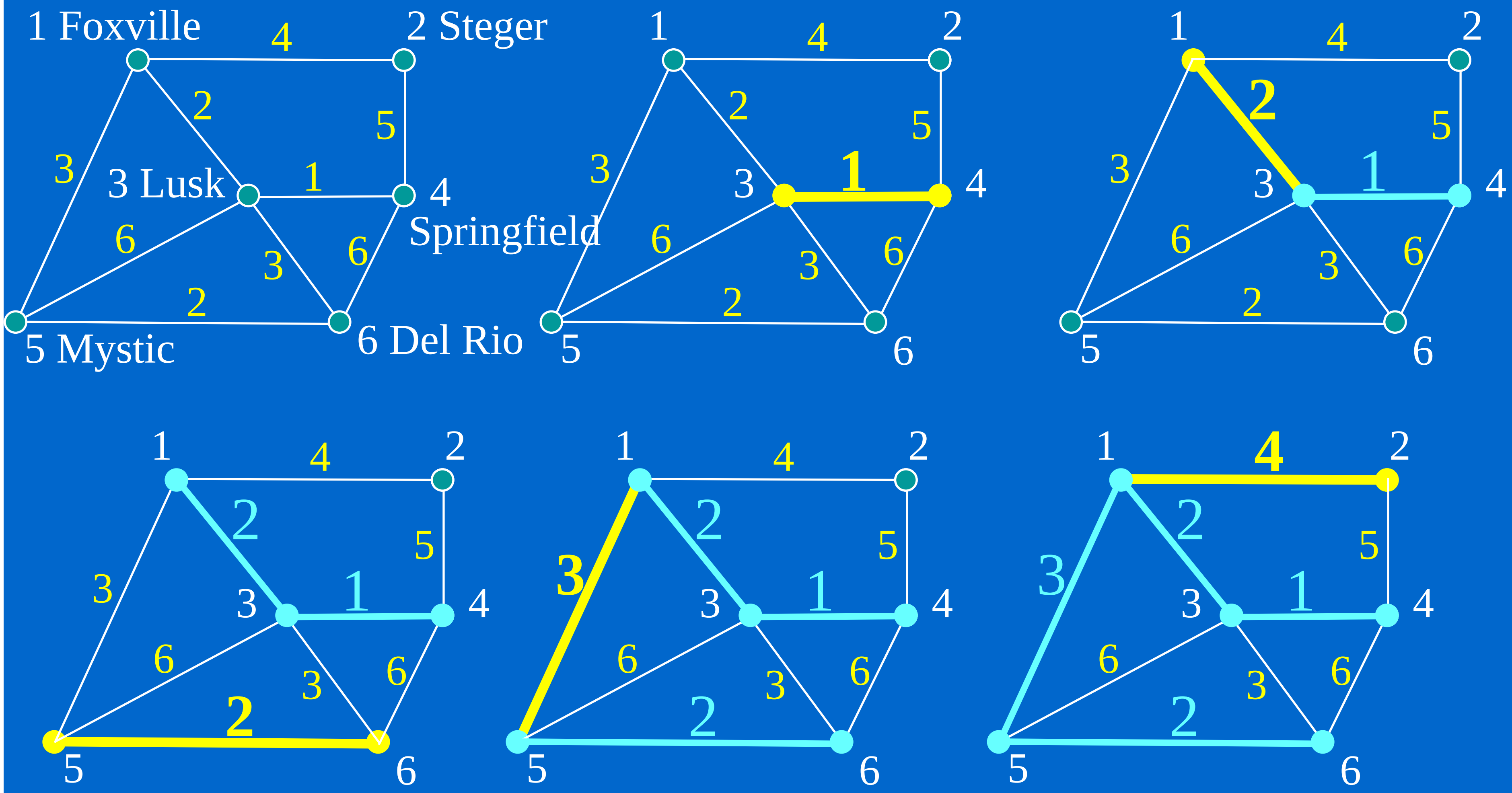
Kruskal's MST (2/11)

✧ Kruskal's algorithm



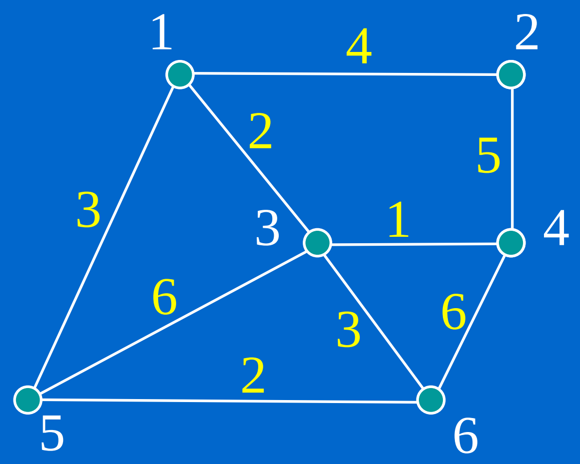
Kruskal's MST (2/11)

✧ Kruskal's algorithm



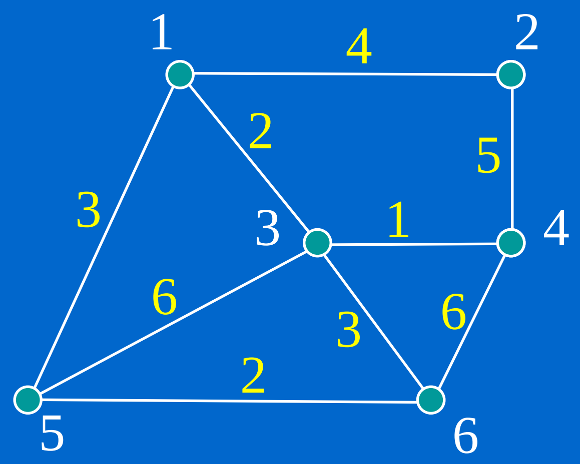
1+2+2+3+4=12

Kruskal's MST (3/11)



❶ find the edge with minimal weight
❷ add to MST if the edge does not form a cycle

Kruskal's MST (3/11)

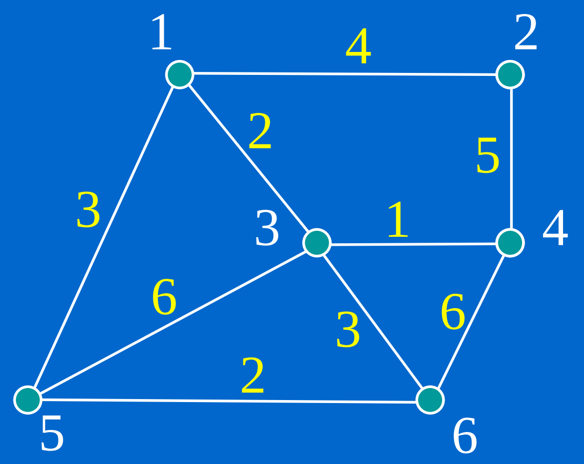


Array of edges:

(1,2,4),(1,3,2),(1,5,3),(2,4,5),(3,4,1),(3,5,6),
(3,6,3),(4,6,6),(5,6,2)

❶ find the edge with minimal weight
❷ add to MST if the edge does not form a cycle

Kruskal's MST (3/11)



Array of edges:

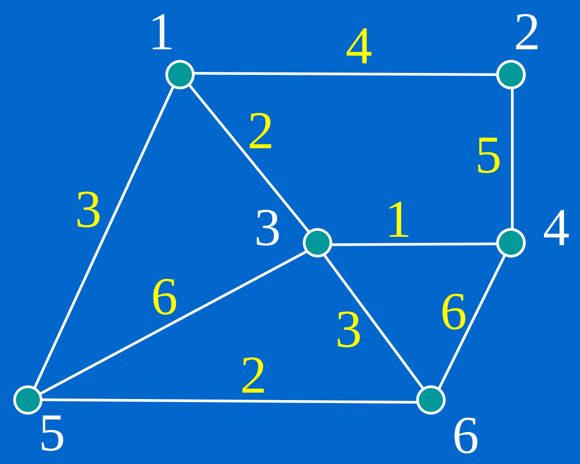
(1,2,4),(1,3,2),(1,5,3),(2,4,5),(3,4,1),(3,5,6),
(3,6,3),(4,6,6),(5,6,2)

Sorted array of edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

❶ find the edge with minimal weight
❷ add to MST if the edge does not form a cycle

Kruskal's MST (3/11)



Array of edges:

(1,2,4),(1,3,2),(1,5,3),(2,4,5),(3,4,1),(3,5,6),
(3,6,3),(4,6,6),(5,6,2)

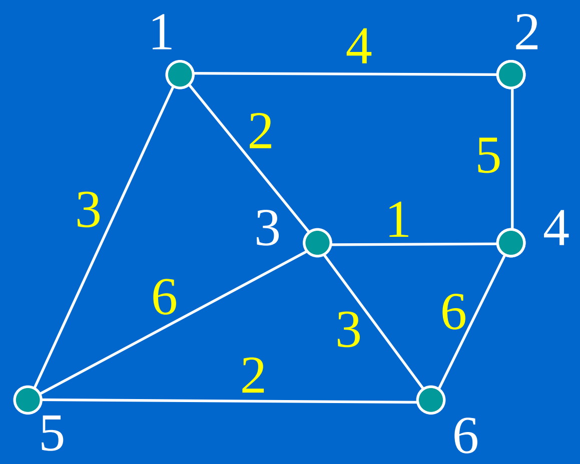
Sorted array of edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

MST:{}

- ❶ find the edge with minimal weight
- ❷ add to MST if the edge does not form a cycle

Kruskal's MST (3/11)



Array of edges:

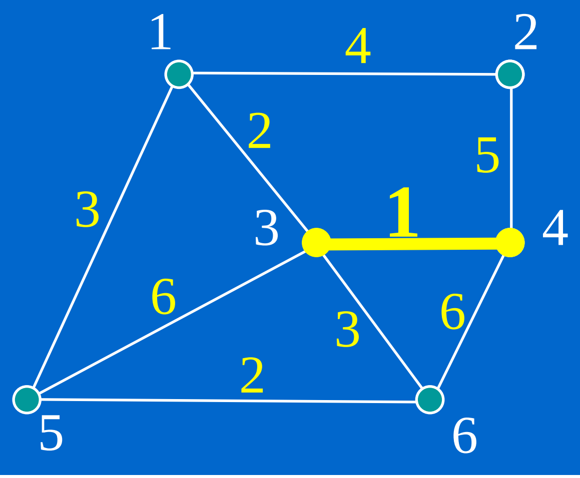
(1,2,4),(1,3,2),(1,5,3),(2,4,5),(3,4,1),(3,5,6),
(3,6,3),(4,6,6),(5,6,2)

Sorted array of edges:

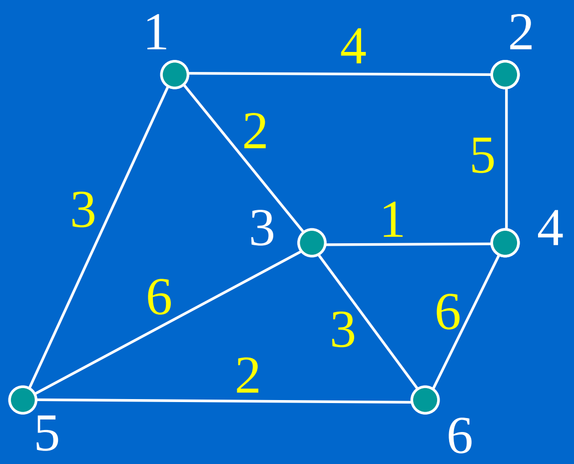
(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

MST:{}

❶ find the edge with minimal weight
❷ add to MST if the edge does not form a cycle



Kruskal's MST (3/11)



Array of edges:

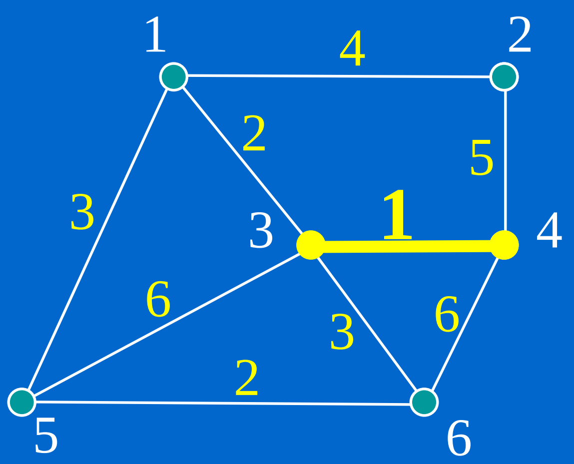
(1,2,4),(1,3,2),(1,5,3),(2,4,5),(3,4,1),(3,5,6),
(3,6,3),(4,6,6),(5,6,2)

Sorted array of edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

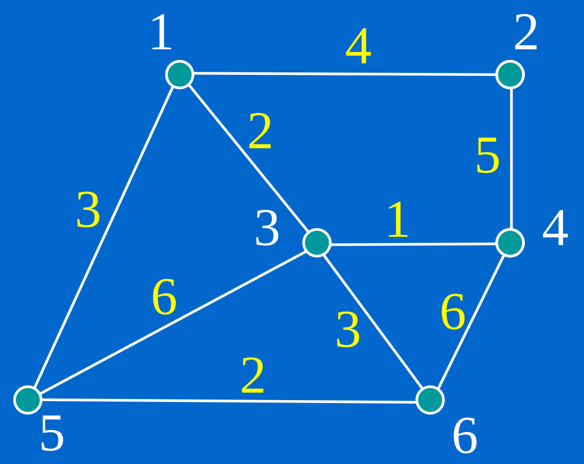
MST:{}

- ❶ find the edge with minimal weight
- ❷ add to MST if the edge does not form a cycle



MST:{**3**,**4**}

Kruskal’s MST (3/11)



Array of edges:

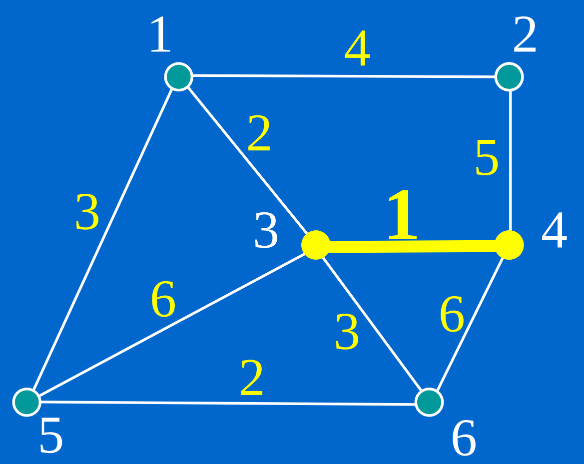
(1,2,4),(1,3,2),(1,5,3),(2,4,5),(3,4,1),(3,5,6),
(3,6,3),(4,6,6),(5,6,2)

Sorted array of edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

MST:{}

❶ find the edge with minimal weight
❷ add to MST if the edge does not form a cycle

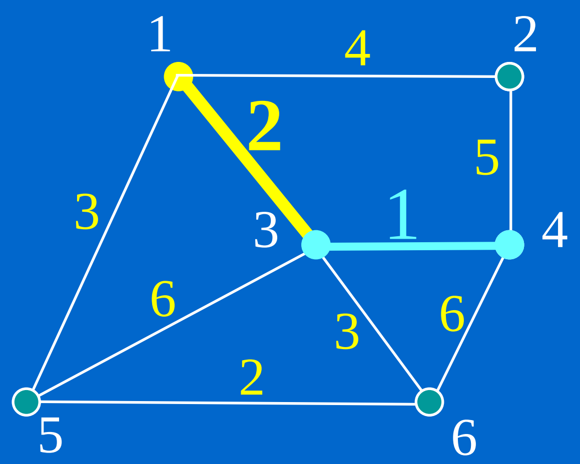


MST:{**3**,**4**}

Remaining edges:

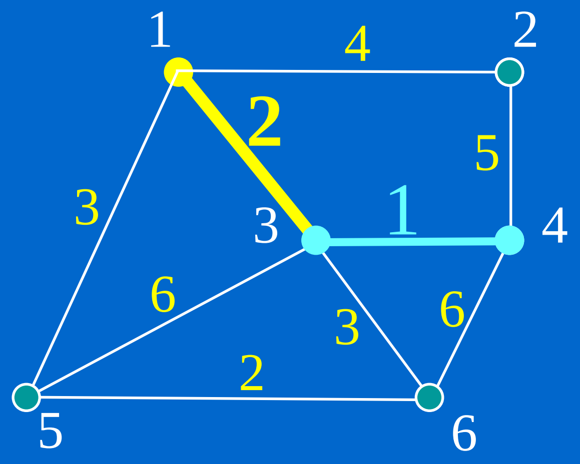
(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

Kruskal's MST (4/11)



MST:{1,3,4}

Kruskal's MST (4/11)

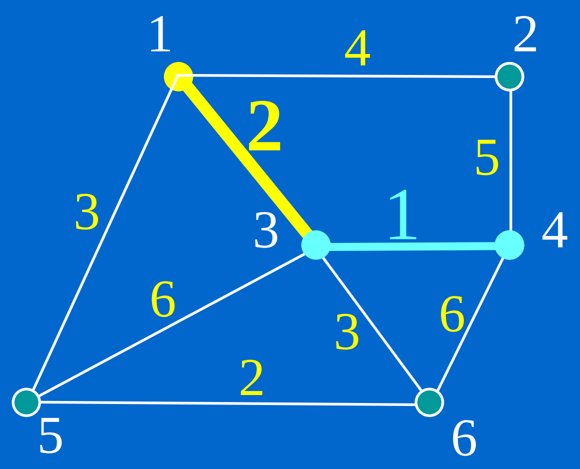


MST:{1,3,4}

Remaining edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),(2,4,5),(3,5,6),(4,6,6)

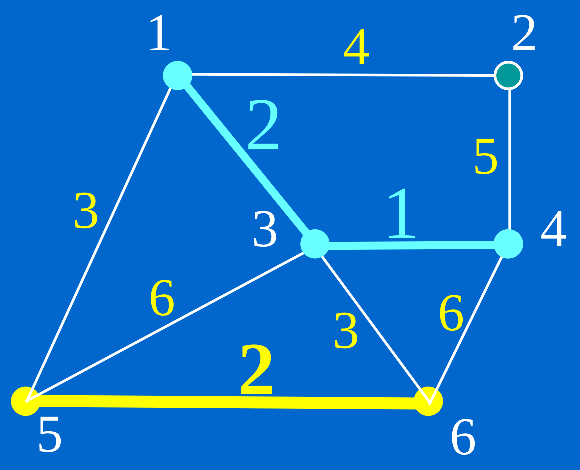
Kruskal's MST (4/11)



MST:{1,3,4}

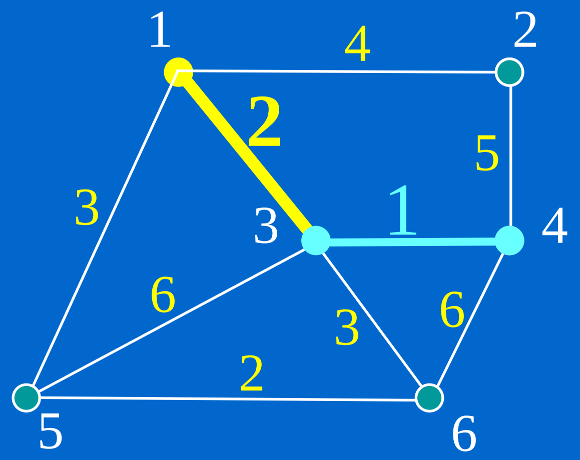
Remaining edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)



MST:{1,3,4},{5,6}

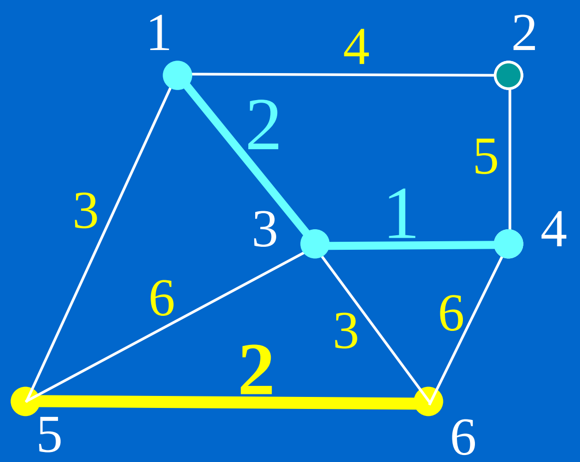
Kruskal’s MST (4/11)



MST:{**1**,3,4}

Remaining edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

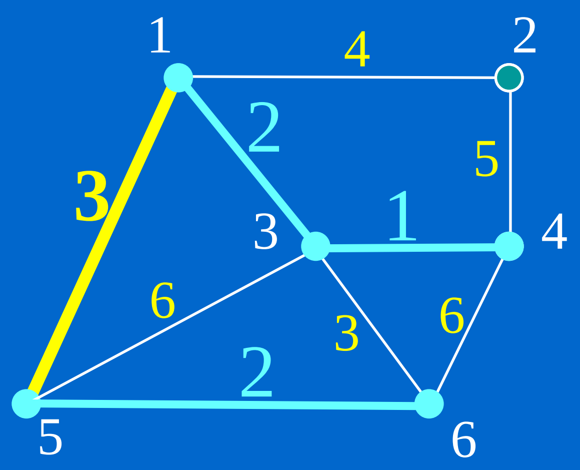


MST:{1,3,4},{**5**,**6**}

Remaining edges:

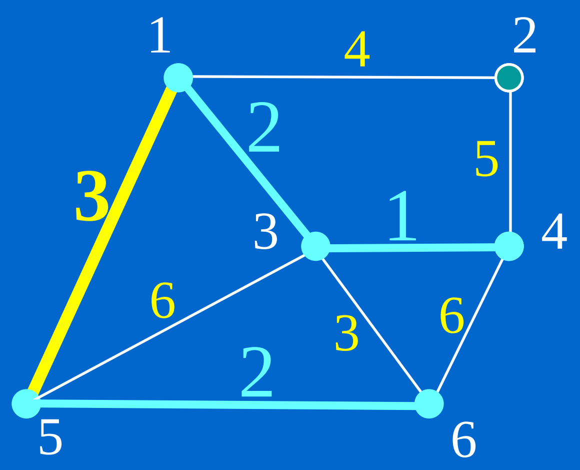
(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

Kruskal's MST (5/11)



MST:{1,3,4,5,6}

Kruskal’s MST (5/11)

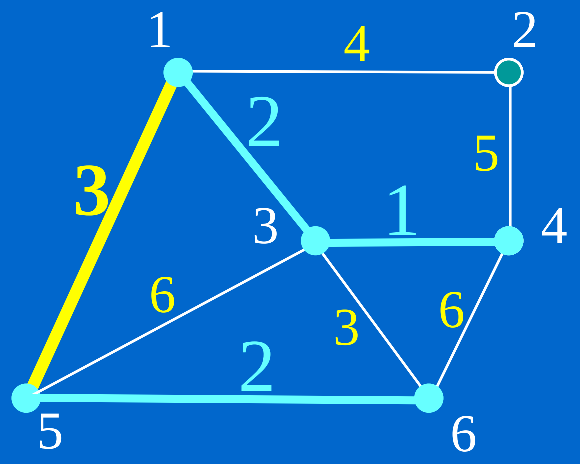


MST:{1,3,4,5,6}

Remaining edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

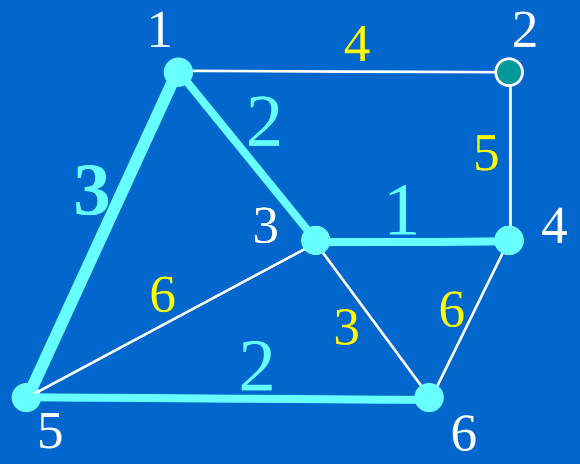
Kruskal's MST (5/11)



MST:{1,3,4,5,6}

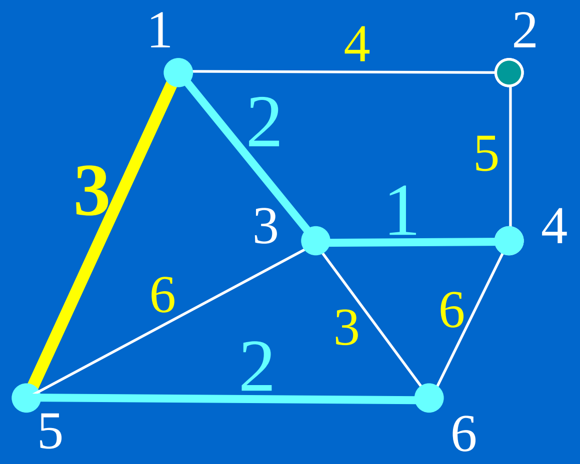
Remaining edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),(2,4,5),(3,5,6),(4,6,6)



MST:{1,3,4,5,6}

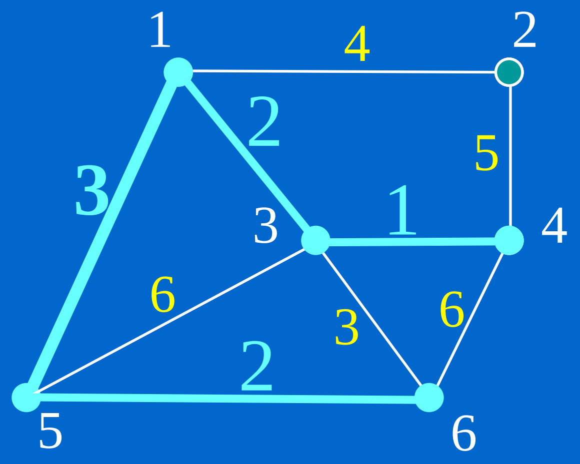
Kruskal's MST (5/11)



MST:{1,3,4,5,6}

Remaining edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),(2,4,5),(3,5,6),(4,6,6)

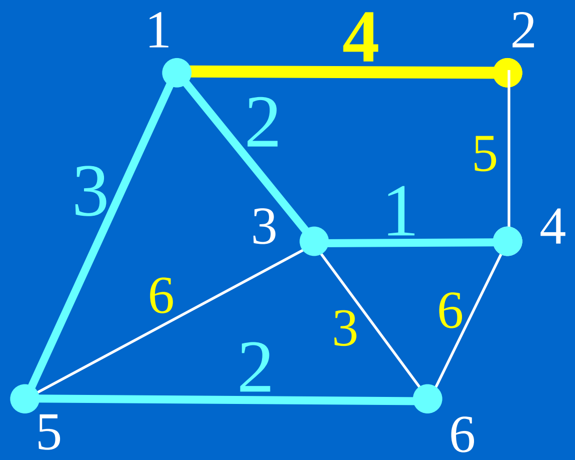


MST:{1,3,4,5,6}

Remaining edges:

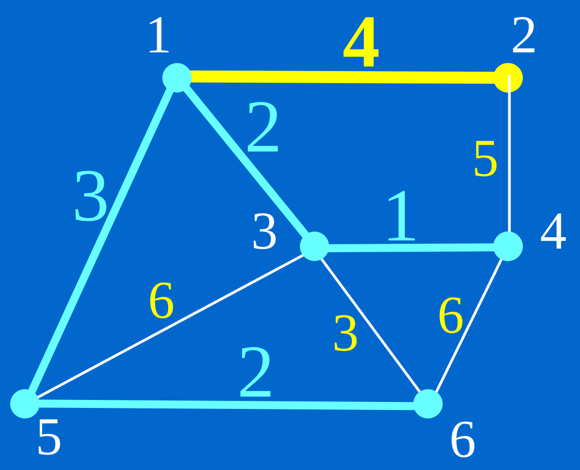
(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),(2,4,5),(3,5,6),(4,6,6)

Kruskal's MST (6/11)



MST:{1,2,3,4,5,6}

Kruskal's MST (6/11)



MST:{1,2,3,4,5,6}

Remaining edges:

(3,4,1),(1,3,2),(5,6,2),(1,5,3),(3,6,3),(1,2,4),
(2,4,5),(3,5,6),(4,6,6)

Kruskal's MST (7/11)

Array of edges: (vertex1, vertex2, weight)

(1,2,4),(1,3,2),(1,5,3),(2,4,5),(3,4,1),(3,5,6),(3,6,3),(4,6,6),(5,6,2)

- Implementation ❶: 2-dimensional arrays (or parallel arrays)

```
int edges[][3] = {{1,2,4},{1,3,2},{1,5,3},{2,4,5},{3,4,1},
                  {3,5,6},{3,6,3},{4,6,6},{5,6,2}};
int nEdges = sizeof(edges) / sizeof(int[3]);
```

- Implementation ❷: array of struct

```
struct Edge {
   int vertex1, vertex2, weight;
};
struct Edge edges[] = {{1,2,4},{1,3,2},{1,5,3},{2,4,5},{3,4,1},
                       {3,5,6},{3,6,3},{4,6,6},{5,6,2}};
int nEdges = sizeof(edges) / sizeof(struct Edge);
```

Kruskal's MST (8/11)

Sorted array of edges:

(3,4,1),(5,6,2),(1,3,2),(1,5,3),(3,6,3),(1,2,4),(2,4,5),(3,5,6),(4,6,6)

- Simple selection sort on 2-dimensional arrays (slightly different results from previous slides)

```
void selectionSort(int edges[][3], int nEdges) {
    int i, max;
    for (i=0; i<nEdges-1; i++) {
        max = findMaximum(edges, nEdges-i);
        swap(edges[nEdges-i-1], edges[max]);
    }
}

int findMaximum(int edges[][3], int nEdges) {
    int i, max=nEdges-1;
    for (i=nEdges-2; i>=0; i--)
        if (edges[i][2] > edges[max][2])
            max = i;
    return max;
}
```

```
void swap(int a[3], int b[3]) {
    int tmp, i;
    for (i=0; i<3; i++) {
        tmp = a[i];
        a[i] = b[i];
        b[i] = tmp;
    }
}
```

Kruskal's MST (9/11)

Sorted array of edges:

(3,4,1),(5,6,2),(1,3,2),(1,5,3),(3,6,3),(1,2,4),(2,4,5),(3,5,6),(4,6,6)

- stdlib qsort on array of structs

```
#include <stdlib.h>

int compare(void *arg1, void *arg2) {
    return ((struct Edge *)arg1)->weight - ((struct Edge *)arg2)->weight;
}

qsort(edgelist, nEdges, sizeof(struct Edge), compare);
```

Sorted array of edges:

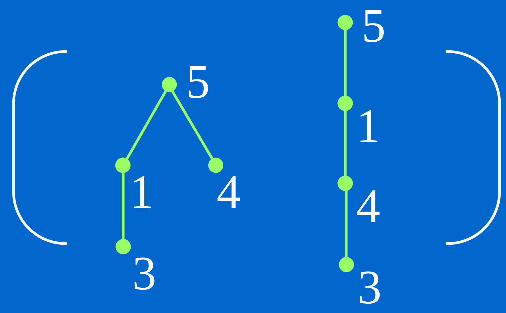
(3,4,1),**(1,3,2),(5,6,2)**,(1,5,3),(3,6,3),(1,2,4),(2,4,5),(3,5,6),(4,6,6)

- requires a stable sorting algorithm: e.g. bubble, bucket, insertion, counting, merge, radix, …

Kruskal's MST (10/11)

MST:{}→{3,4}→{1,3,4}→{1,3,4},{5,6}→{1,3,4,5,6}→{1,2,3,4,5,6}

- Require "set processing" tools: union, comparison
- Specially, these are disjoint sets (Section 3.6 of JohnsonBaugh, pp.150):
  - Set members are held in the same tree, root node represents the set
  - use an array ***parent*** to implement the set membership and provide three interfaces:
    - **makeset**(i): construct the set {i}
    - **findset**(i): returns the representative node of the set
    - **union**(i,j): joins the set containing i and the set containing j



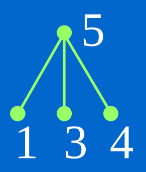

```
void makeset(int i, int nNodes, int parent[]) {
   if ((i<0)||(i>=nNodes)) return;
   parent[i] = i;
}
```

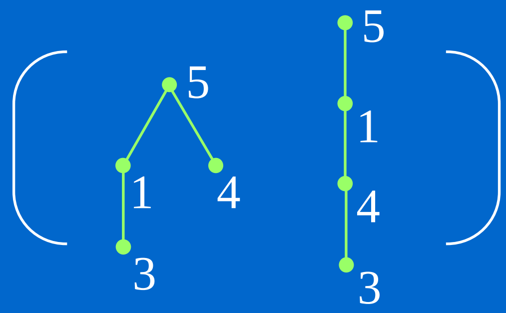
```
int findset(int i, int nNodes, int parent[]) {
   if ((i<0)||(i>=nNodes)) return -1;
   while (i != parent[i])
      i = parent[i];
   return i;
}
```

Kruskal's MST (10/11)

MST:{}→{3,4}→{1,3,4}→{1,3,4},{5,6}→{1,3,4,5,6}→{1,2,3,4,5,6}

- Require "set processing" tools: union, comparison
- Specially, these are disjoint sets (Section 3.6 of JohnsonBaugh, pp.150):
  - Set members are held in the same tree, root node represents the set
  - use an array ***parent*** to implement the set membership and provide three interfaces:
    - **makeset**(i): construct the set {i}
    - **findset**(i): returns the representative node of the set
    - **union**(i,j): joins the set containing i and the set containing j





parent

1	2	3	4	5	6
5		1	5	5	

```
void makeset(int i, int nNodes, int parent[]) {
    if ((i<0)||(i>=nNodes)) return;
    parent[i] = i;
}
```

```
int findset(int i, int nNodes, int parent[]) {
    if ((i<0)||(i>=nNodes)) return -1;
    while (i != parent[i])
        i = parent[i];
    return i;
}
```

Kruskal's MST (11/11)

```
void mergetrees(int i, int j, int nNodes, int parent[]) {
   if (((i<0)||(i>=nNodes)) || ((j<0)||(j>=nNodes))) return;
   parent[i] = j;
}
void union(int i, int j, int nNodes, int parent[]) {
   if (((i<0)||(i>=nNodes)) || ((j<0)||(j>=nNodes))) return;
   mergetrees(findset(i, nNodes, parent), findset(j, nNodes, parent), nNodes, parent);
}
```

❶ find the edge with minimal weight
❷ add to MST if the edge does not form a cycle

```
for (iEdge=0,treeSize=0; treeSize<nNodes; iEdge++) {
      if (findset(edgelist[iEdge][0], nNodes, nodeSet) !=
          findset(edgelist[iEdge][1], nNodes, nodeSet)) {
         totalWeight = totalWeight + edgelist[iEdge][2]; treeSize++;
         union(edgelist[iEdge][0], edgelist[iEdge][1], nNodes, nodeSet);
      }
}
```