اصول پردازش تصویر

Principles of Image Processing

مصطفی کمالی تبریزی

۲۴ آبان ۱۳۹۹

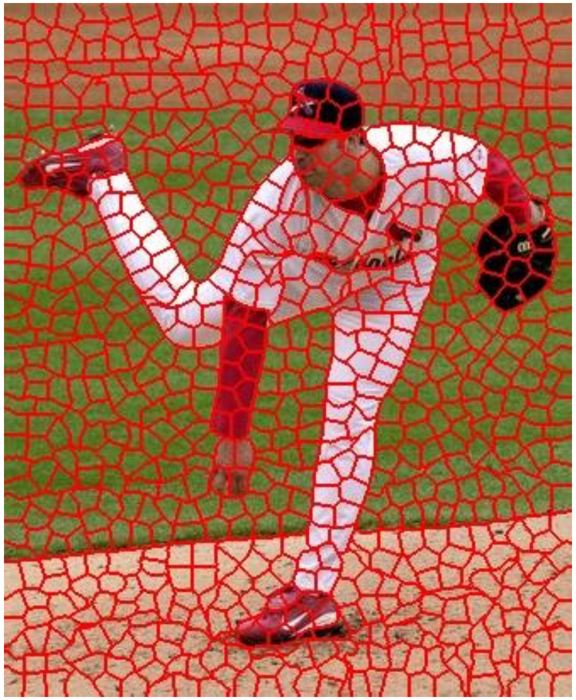
جلسه شانزدهم

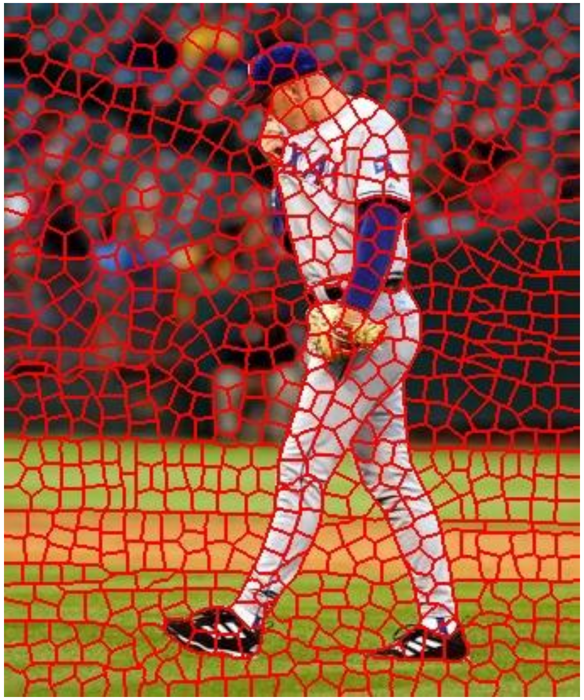
Oversegmentation & Superpixels

Oversegmentation

- Group together similar-looking pixels for efficiency of further processing
  - "Bottom-up" process
  - Unsupervised

"superpixels"





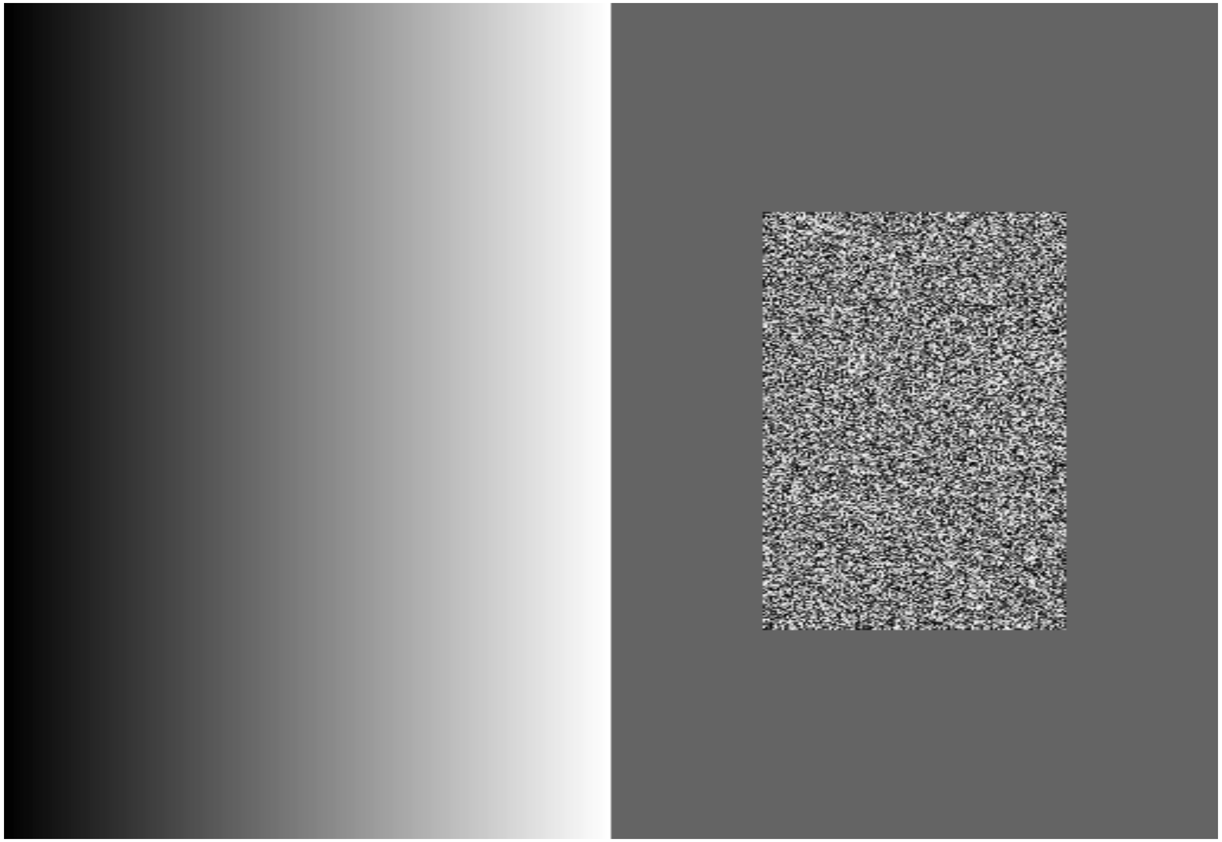
Xiaofeng Ren and Jitendra Malik
Learning a classification model for segmentation,
International Conference on Computer Vision (ICCV), 2003.

Slide Credit: Svetlana Lazebnik

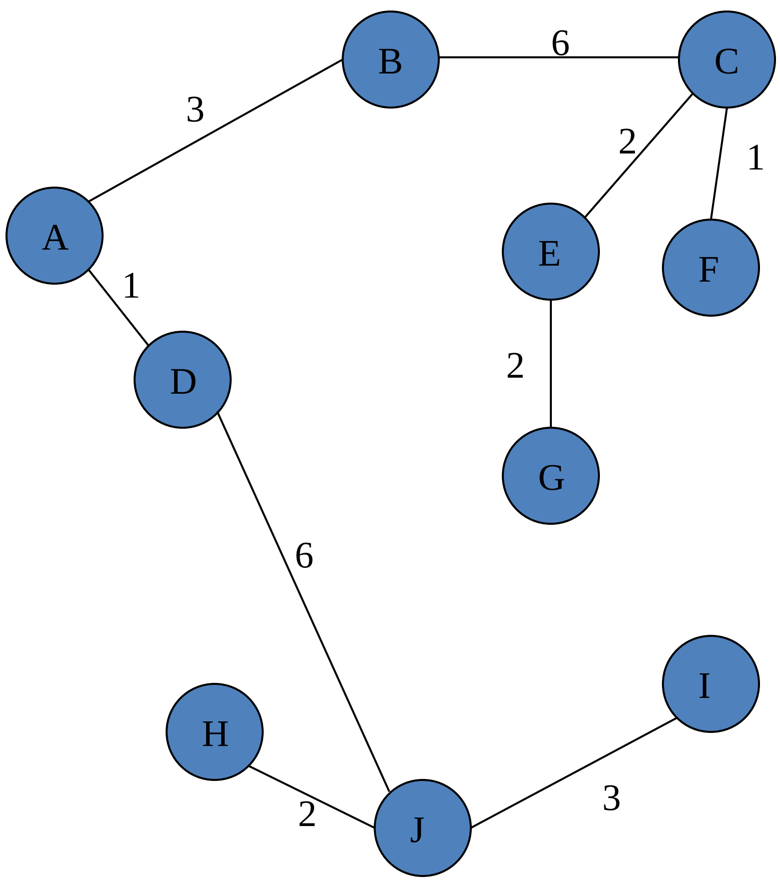
Graph-Based Oversegmentation

Pedro F. Felzenszwalb and Daniel P. Huttenlocher
Efficient Graph-Based Image Segmentation
International Journal of Computer Vision (IJCV), 2004.

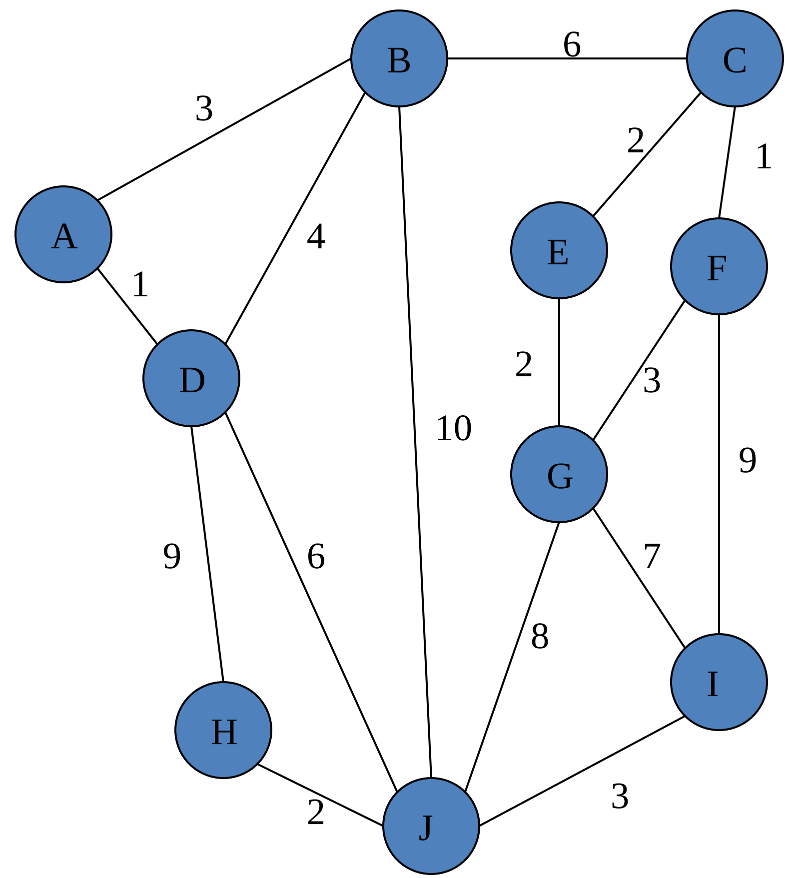
Challenging Example



Minimum Spanning Tree
(Kruskal Algorithm)

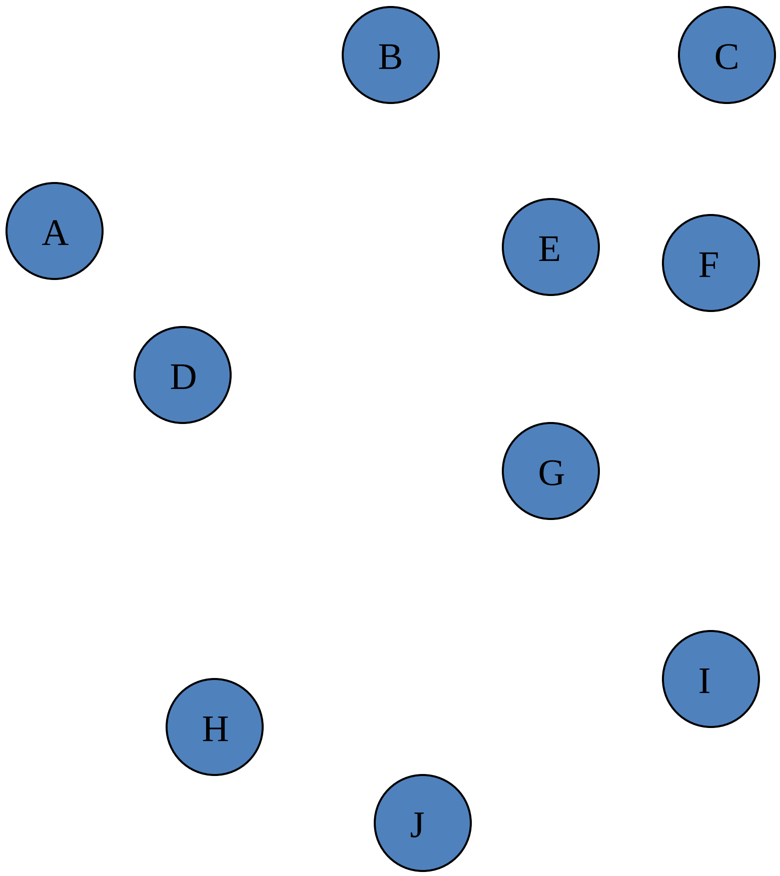


Complete Graph

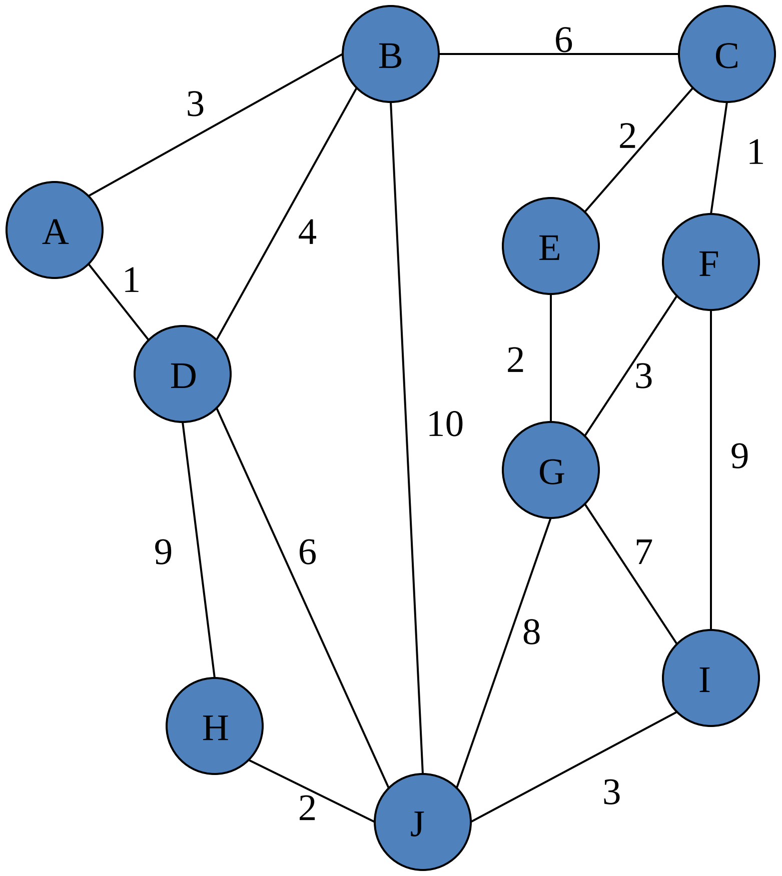


Minimum Spanning Tree

(Kruskal Algorithm)

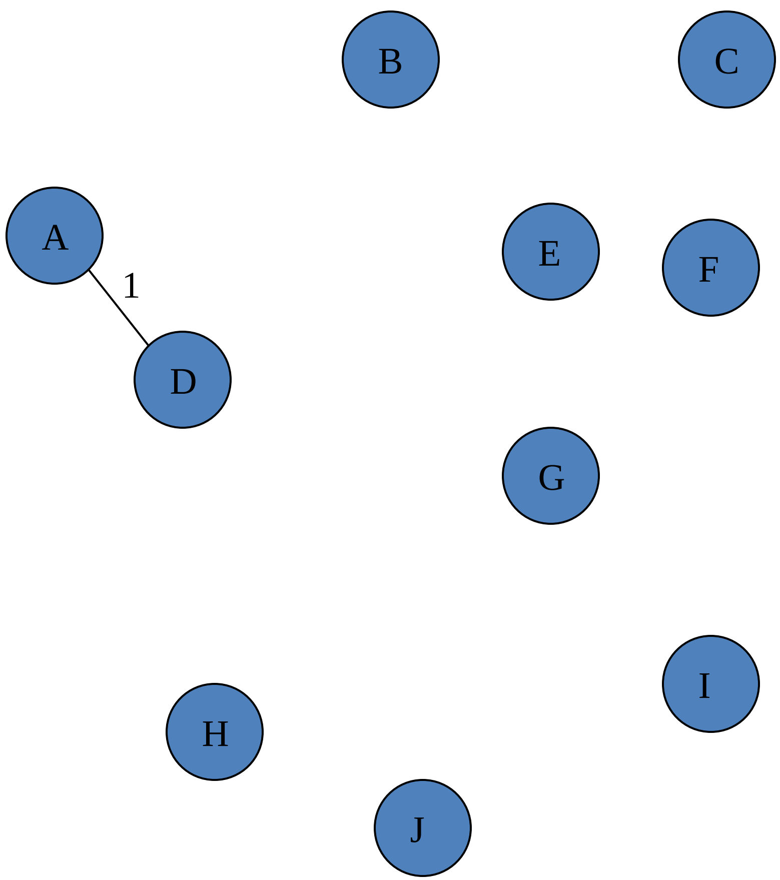


Complete Graph

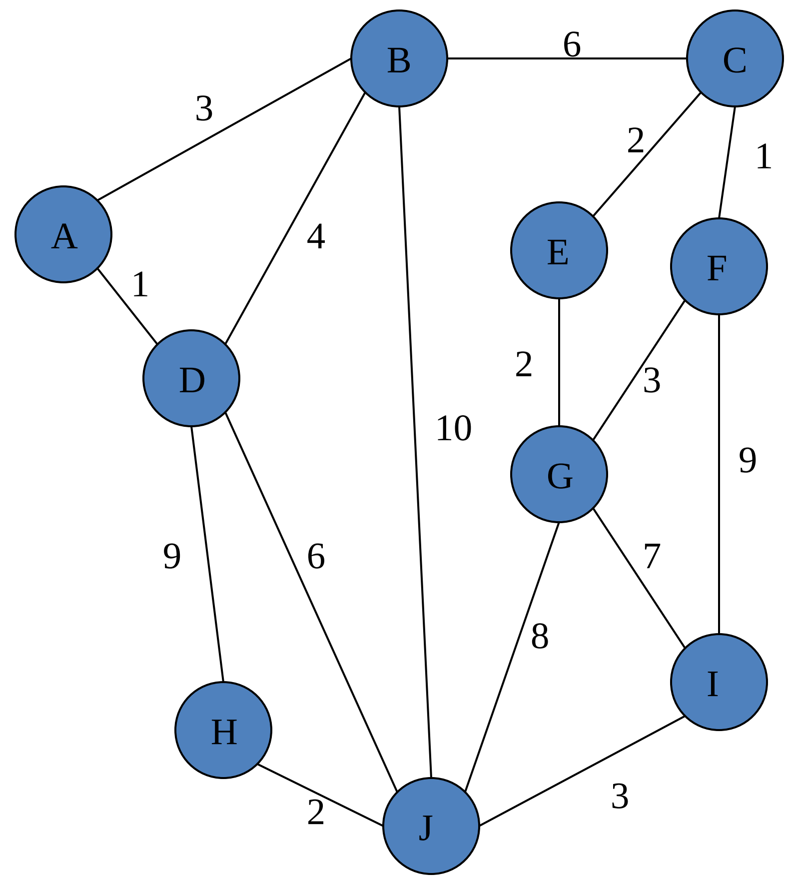


Minimum Spanning Tree

(Kruskal Algorithm)

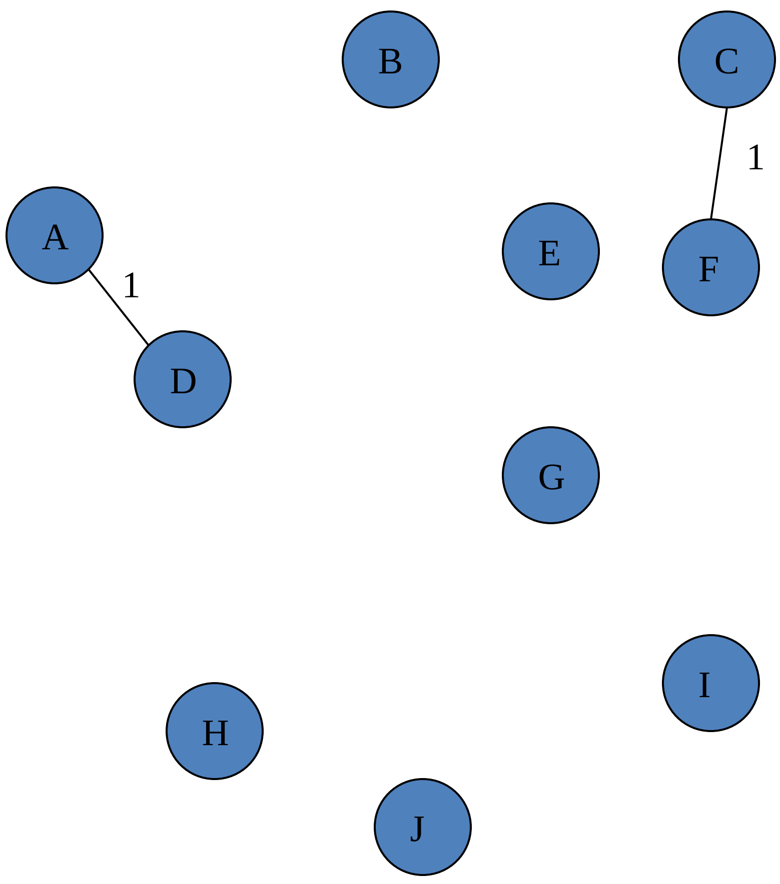


Complete Graph

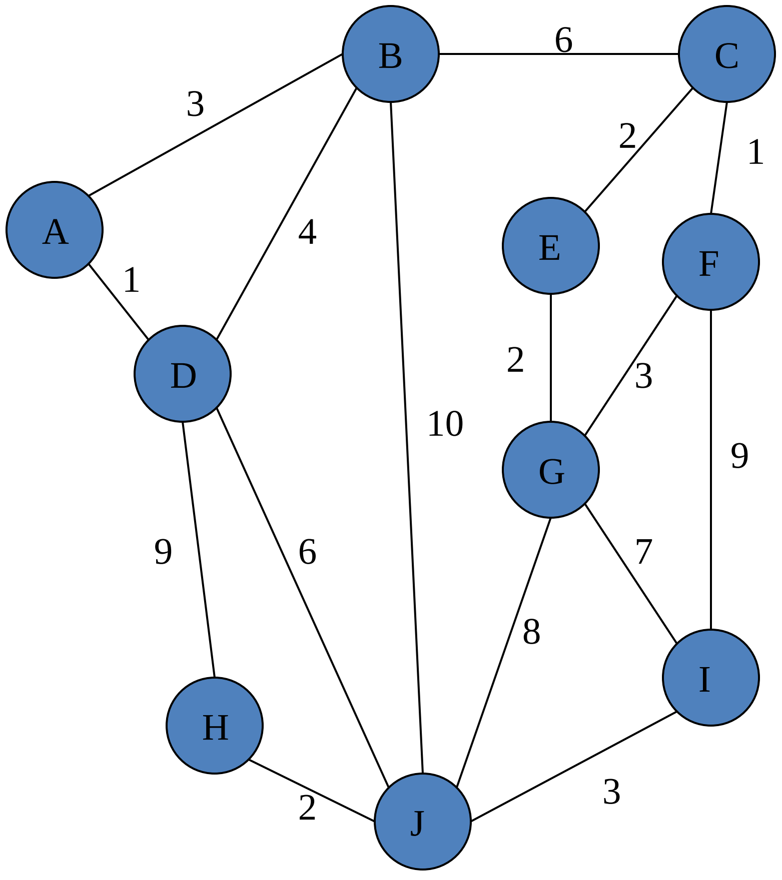


Minimum Spanning Tree

(Kruskal Algorithm)

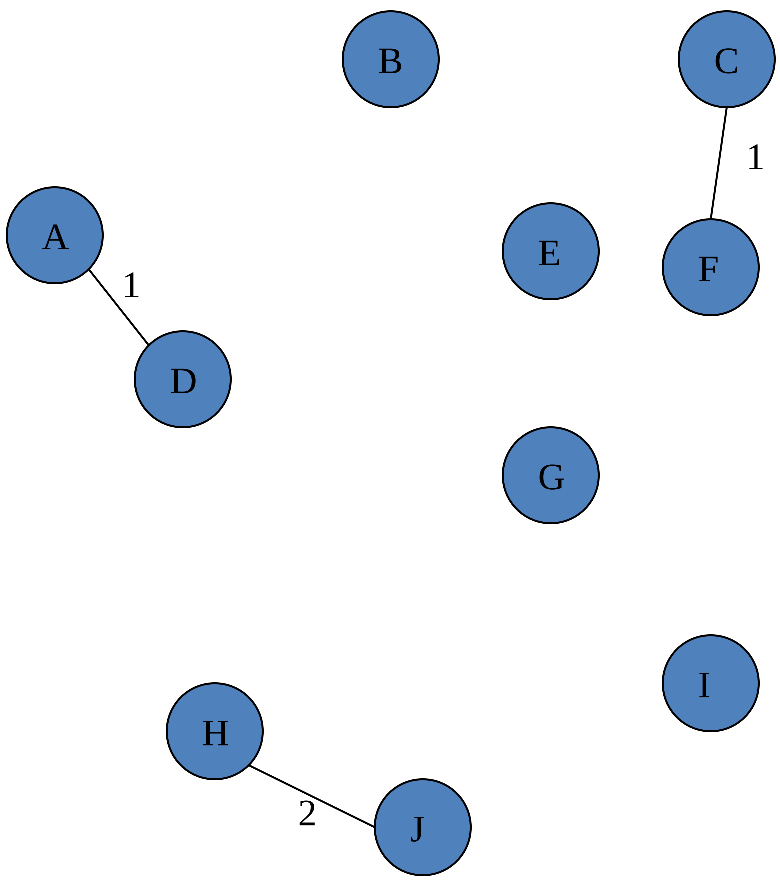


Complete Graph

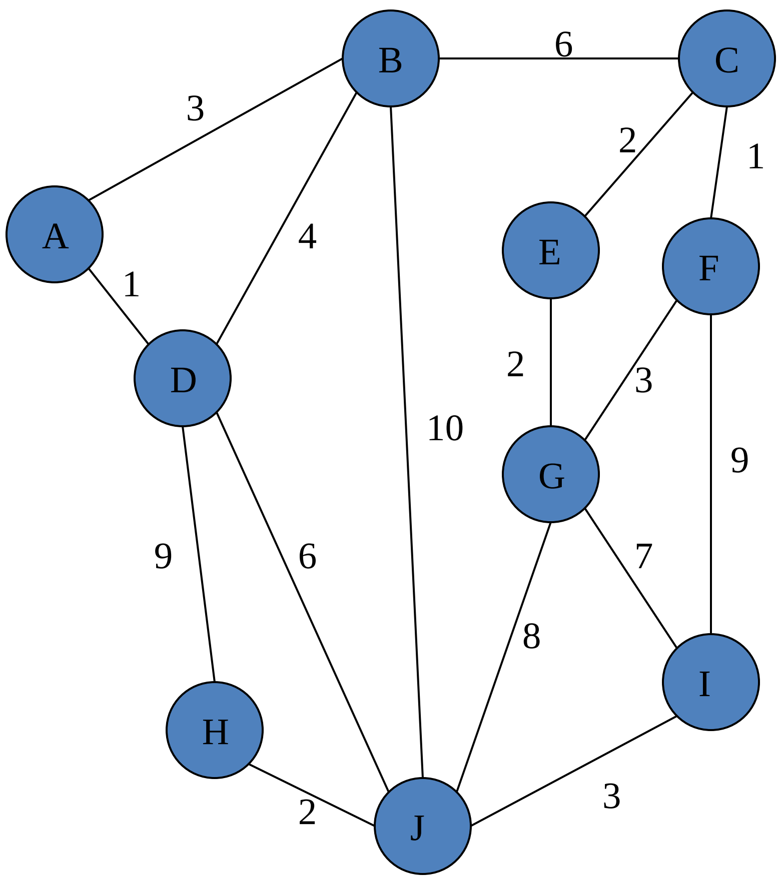


Minimum Spanning Tree

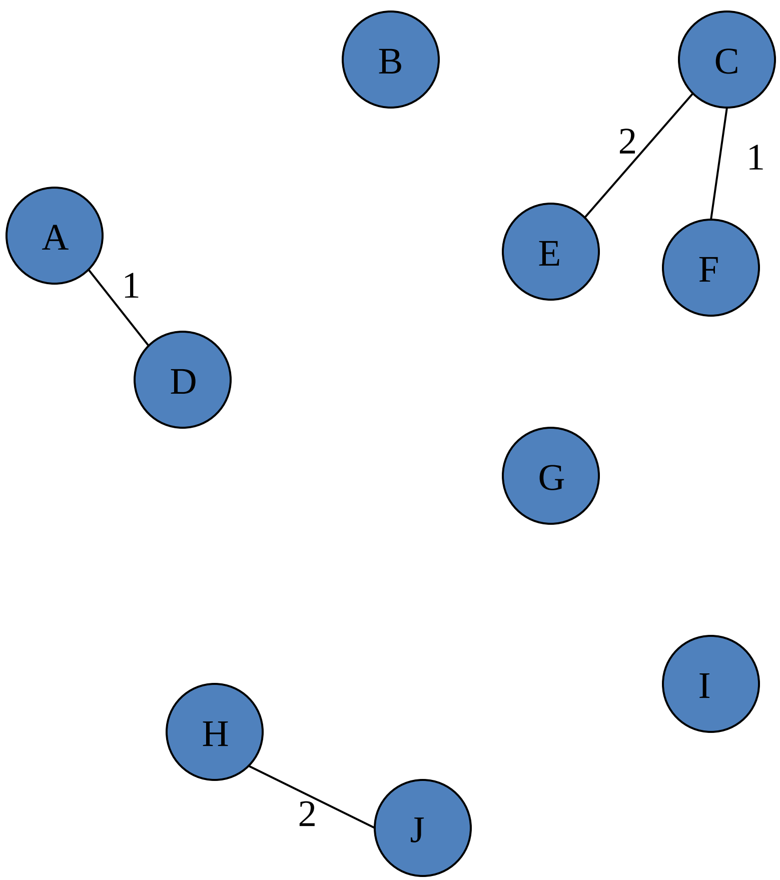
(Kruskal Algorithm)



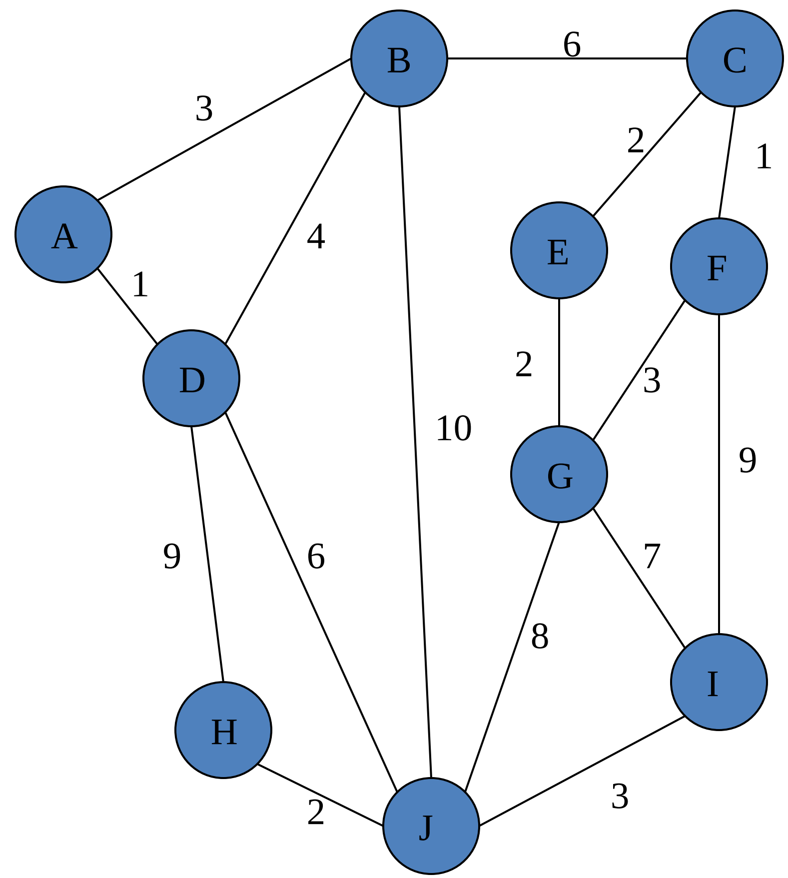
Complete Graph



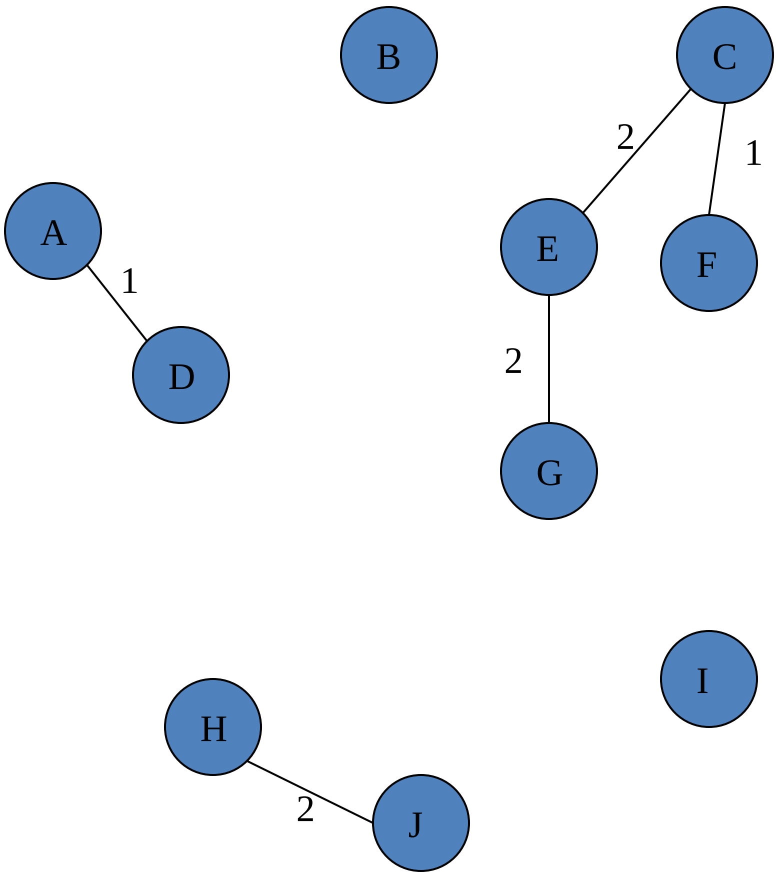
Minimum Spanning Tree (Kruskal Algorithm)



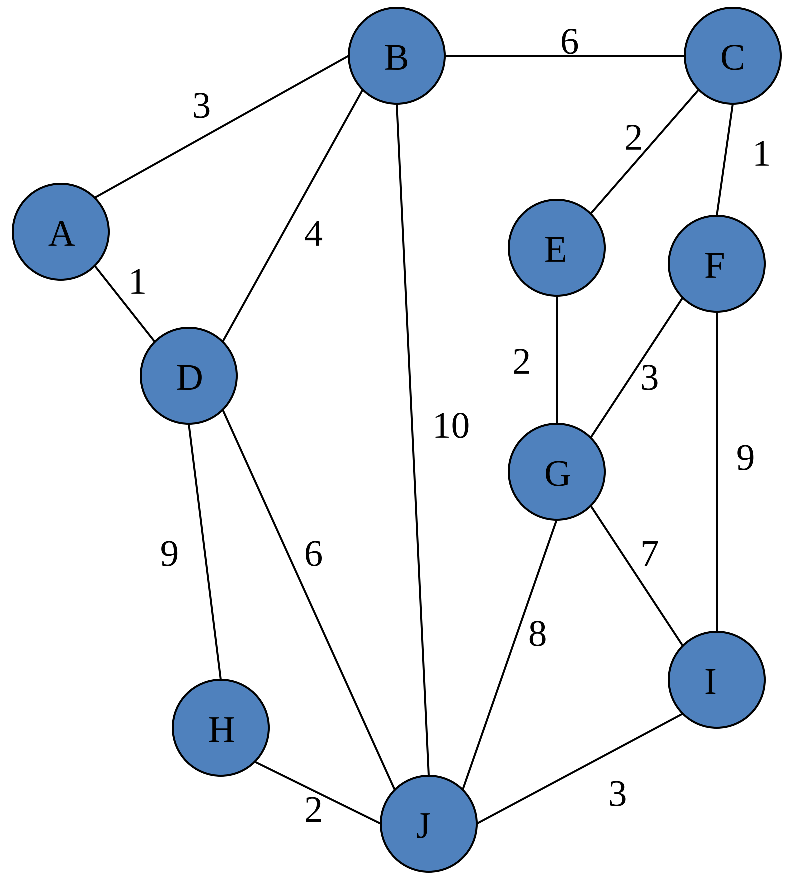
Complete Graph



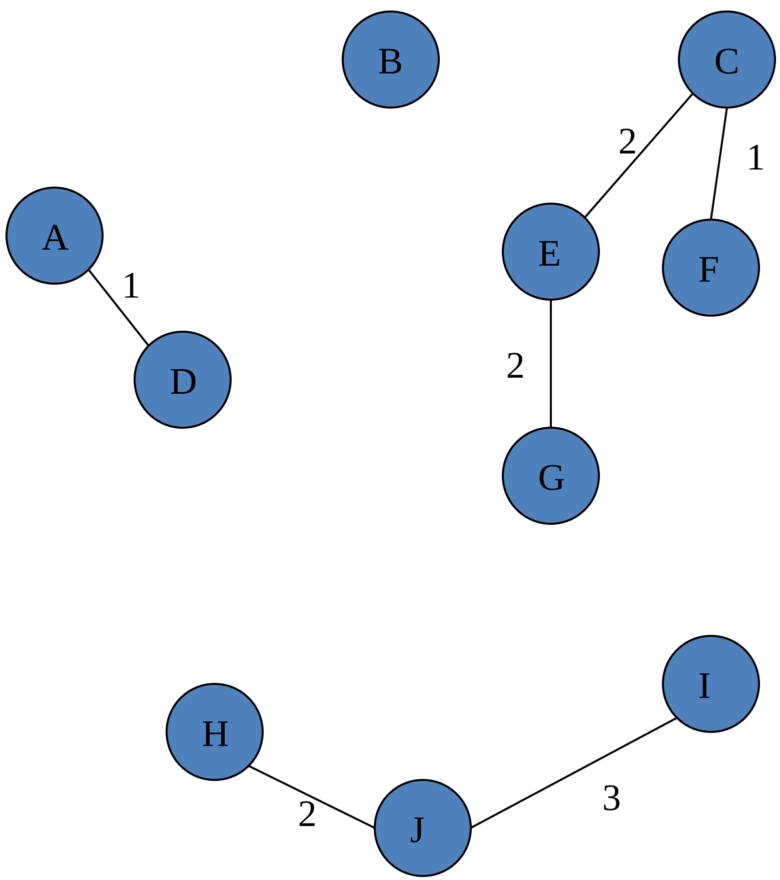
Minimum Spanning Tree (Kruskal Algorithm)



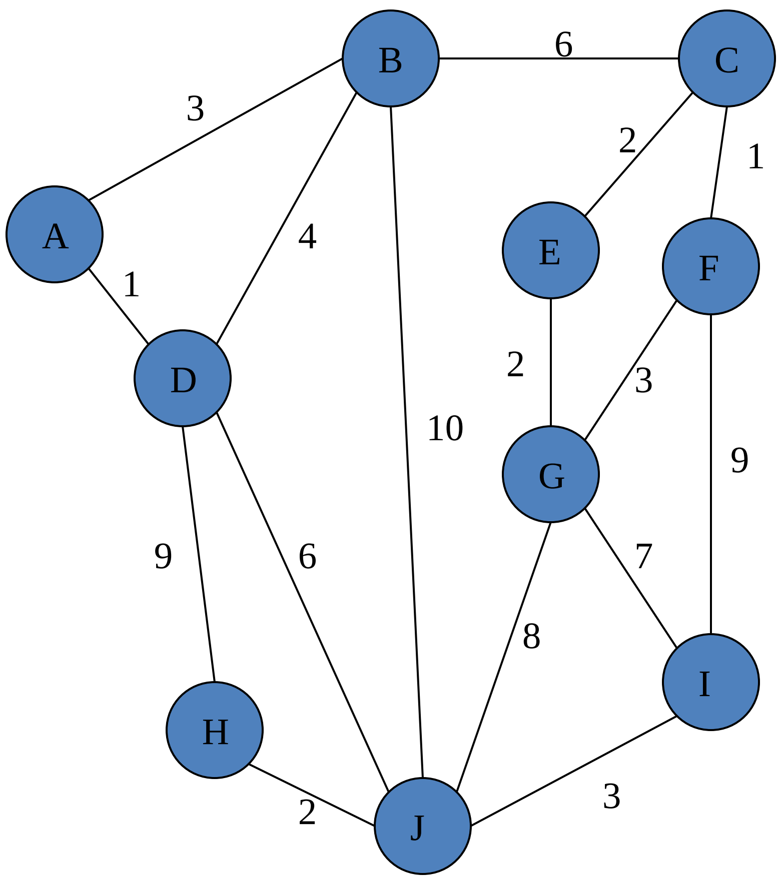
Complete Graph



Minimum Spanning Tree (Kruskal Algorithm)

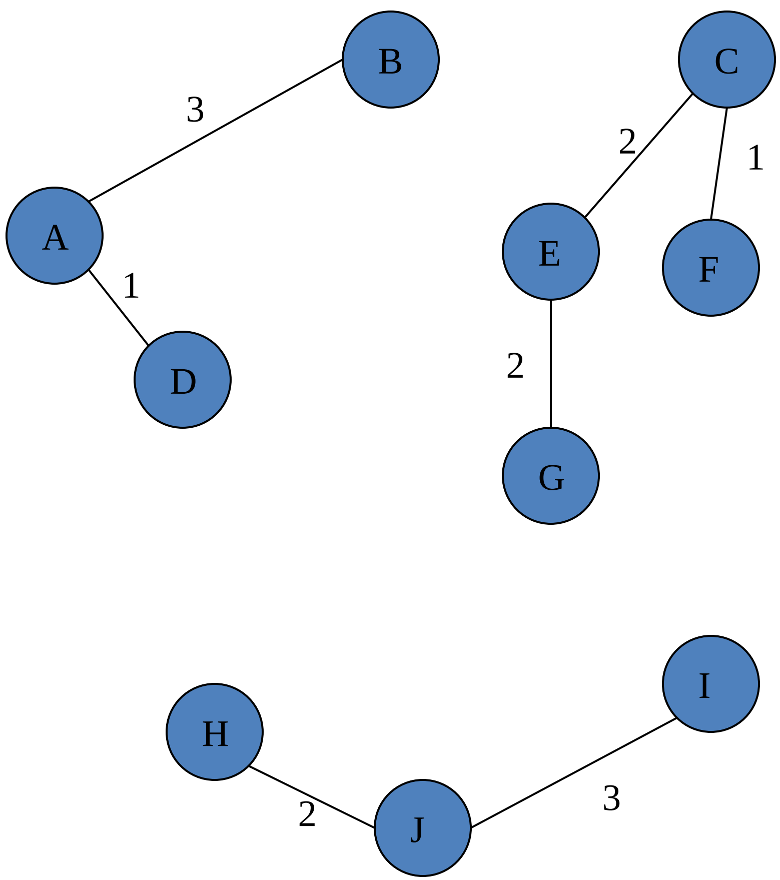


Complete Graph

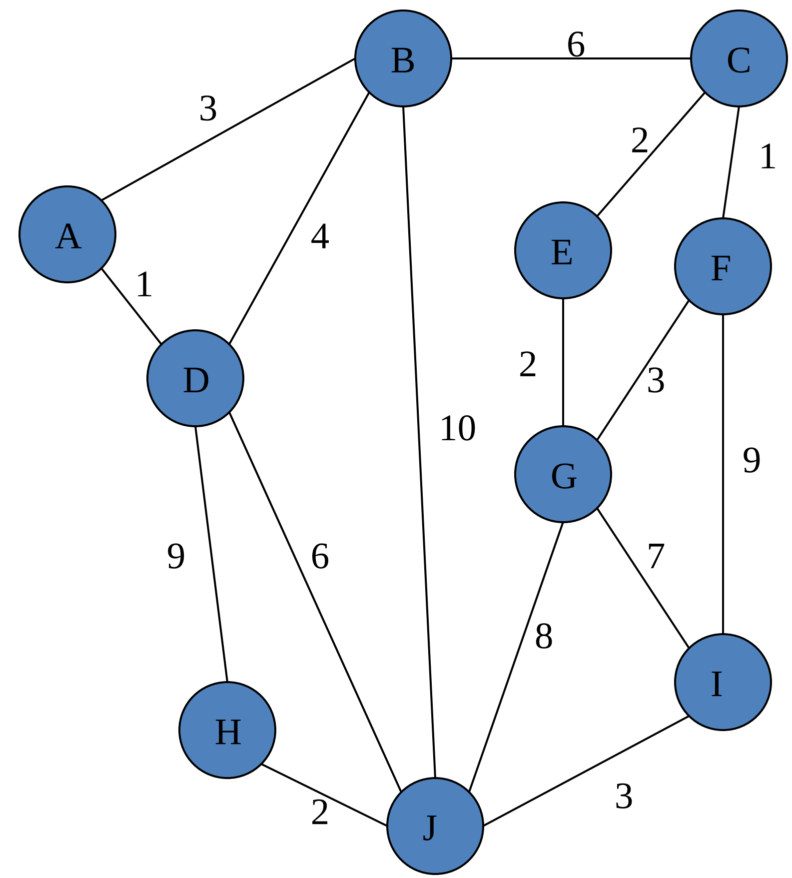


Minimum Spanning Tree

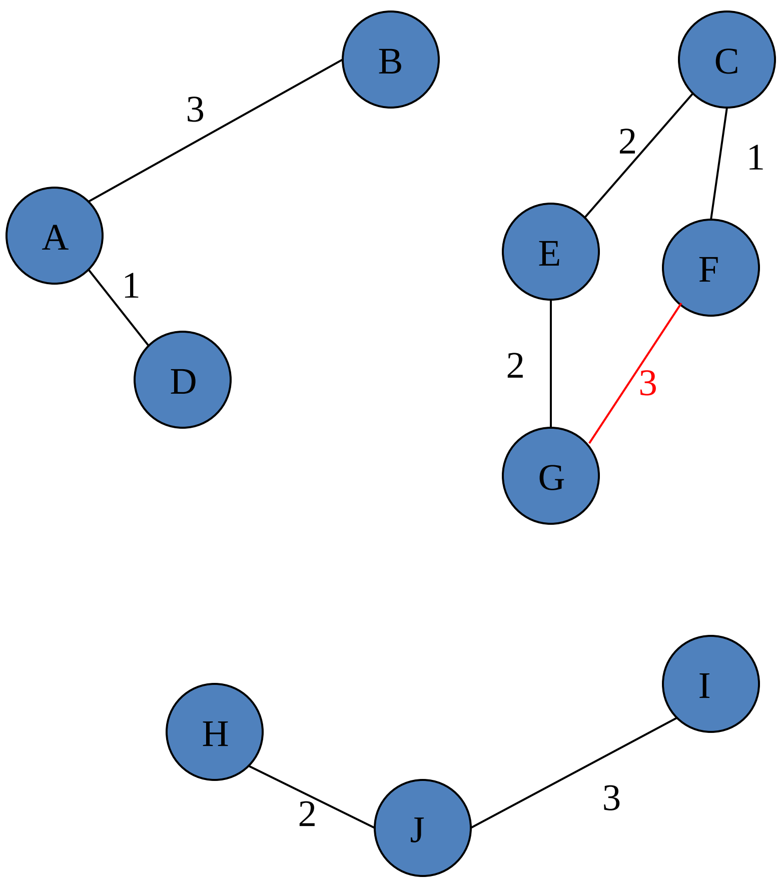
(Kruskal Algorithm)



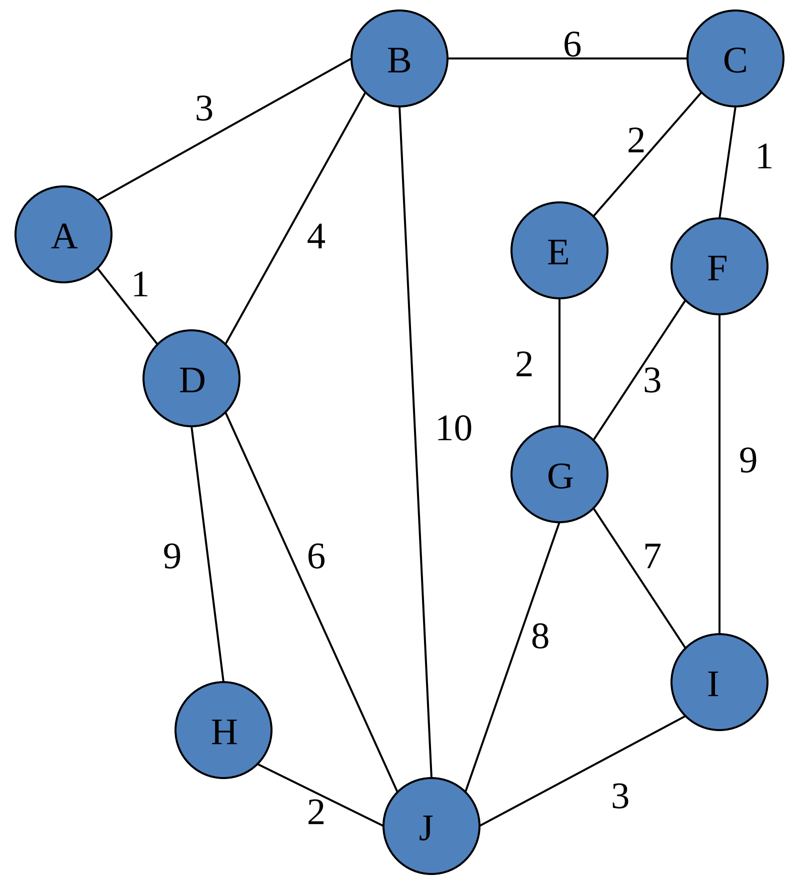
Complete Graph



Minimum Spanning Tree (Kruskal Algorithm)

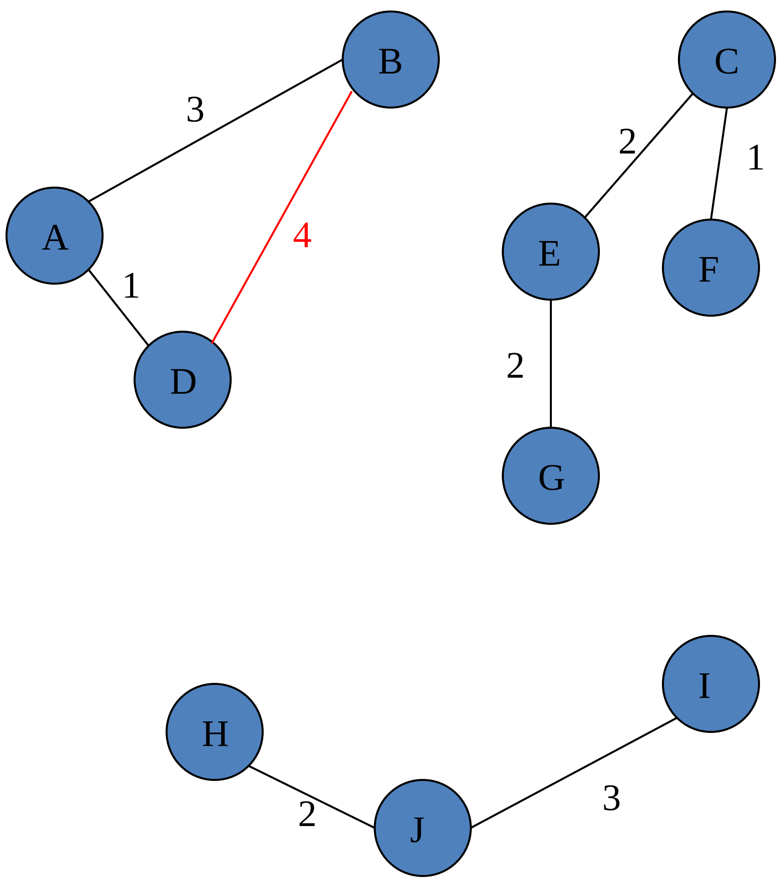


Complete Graph

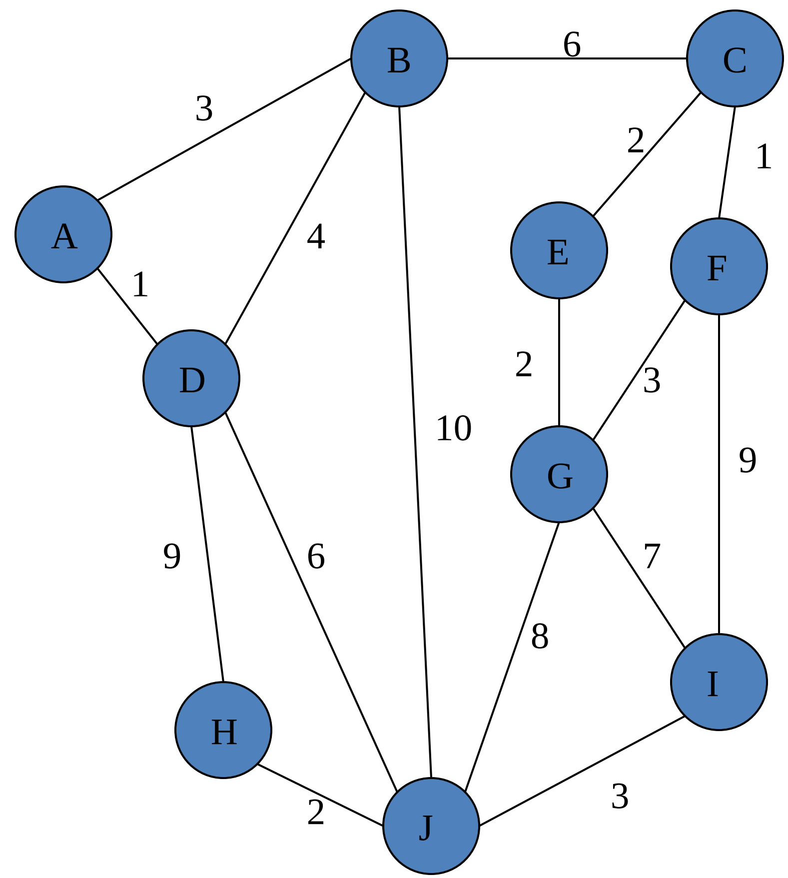


Minimum Spanning Tree

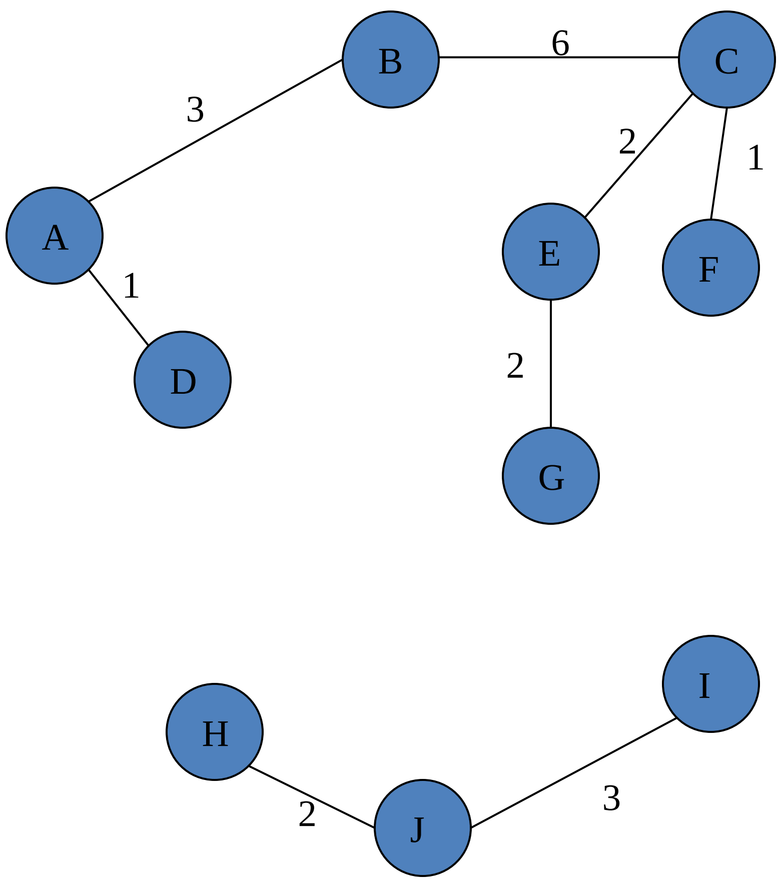
(Kruskal Algorithm)



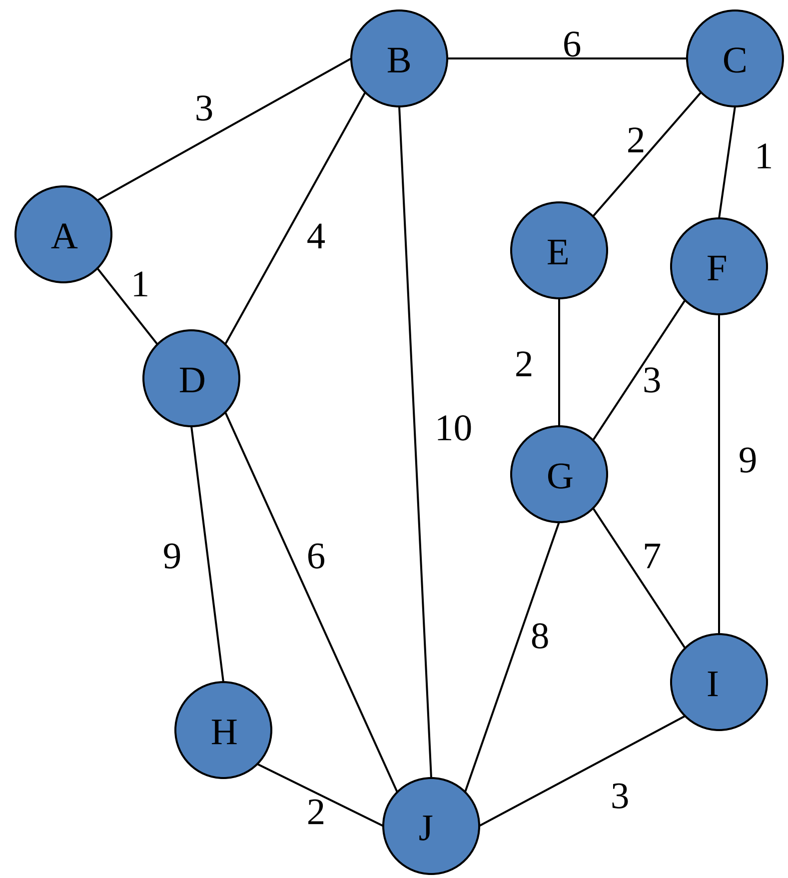
Complete Graph



Minimum Spanning Tree (Kruskal Algorithm)

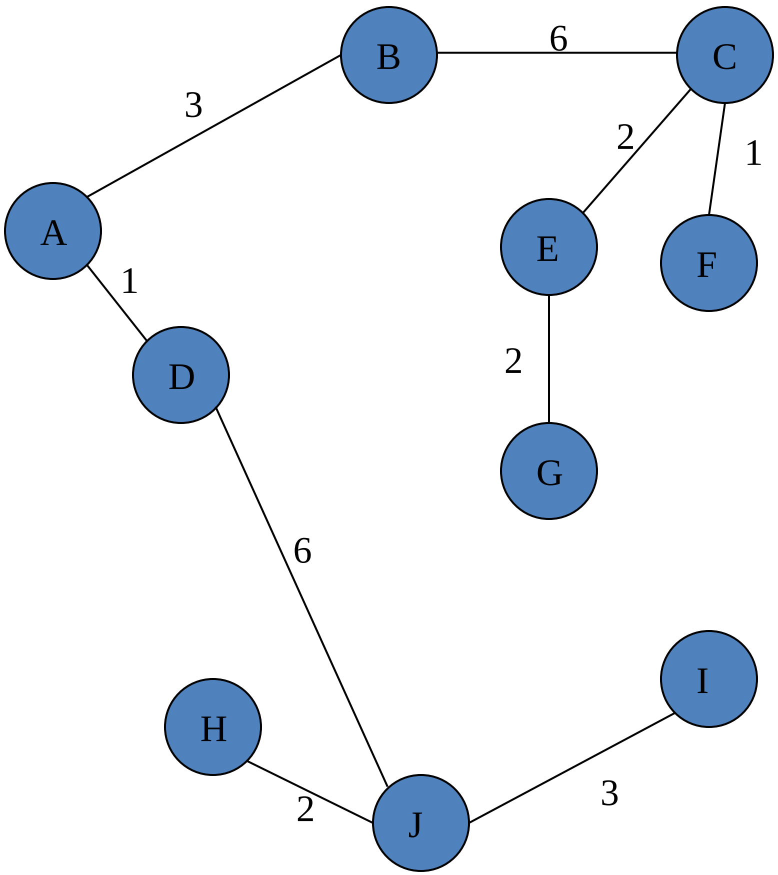


Complete Graph

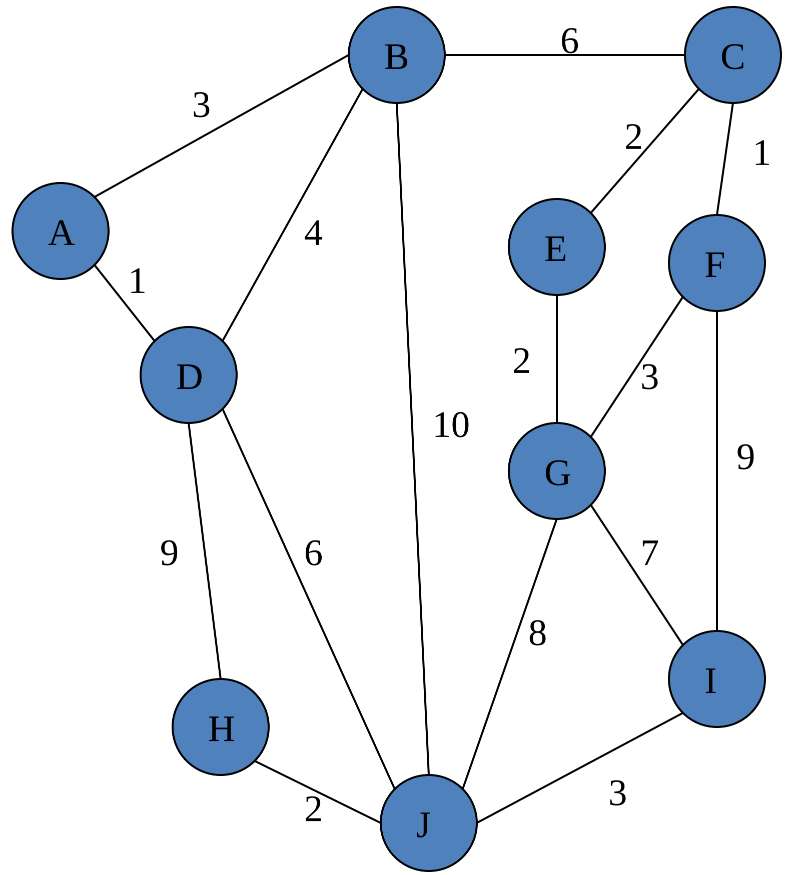


Minimum Spanning Tree

(Kruskal Algorithm)



Complete Graph



Efficient Graph-Based Image Segmentation (Felzenswalb & Huttenlocher)

Consider undirected graph $G = (V\ , E)$ with characteristics below:

- Vertices $v_i \in V$ are descriptors of pixels p_i
- Edges $\left(v_i, v_j\right) \in E$ pair neighboring vertices
- Weights $w\left(\left(v_i, v_j\right)\right)$ are non-negative measures of dissimilarity between neighboring elements v_i and v_j.

Segmentation $S = \{C_1, C_2, \dots, C_n\}$ is a partition of V so that:

- Each C_i is a connected component in graph $G_i = (V, E_i)$
- $E_i \subseteq E$

Efficient Graph-Based Image Segmentation (Felzenswalb & Huttenlocher)

Objective:

Finding a segmentation so that:
elements in a component to be similar, and
elements in different components to be dissimilar.

In other words:
Low weights of edges in a component, and
High weights of edges between two different components.

Efficient Graph-Based Image Segmentation (Felzenswalb & Huttenlocher)

Internal difference of a component $C \subseteq V$:

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

The largest weight in the minimum spanning tree of C, $MST(C, E)$.

Difference between two components $C_1, C_2 \subseteq V$:

$$Dif(C_1, C_2) = \min_{\substack{v_i \in C_1,\ v_j \in C_2 \\ (v_i, v_j) \in E}} w\left((v_i, v_j)\right)$$

The minimum weight edge connecting two components.

Minimum internal difference of two components C_1 and C_2:

$$MInt(C_1, C_2) = \min(\, Int(C_1) + \tau(C_1)\, ,\, Int(C_2) + \tau(C_2)\,)$$

Efficient Graph-Based Image Segmentation (Felzenswalb & Huttenlocher)

Pairwise comparison predicate:

$$D(C_1, C_2) = \begin{cases} true & Dif(C_1, C_2) < MInt(C_1, C_2) \\ false & otherwise \end{cases}$$

True = Merging two components

False = Not merging two components

Algorithm $\boxed{n log(n) \ where \ n \ is \ the \ number \ of \ pixels}$

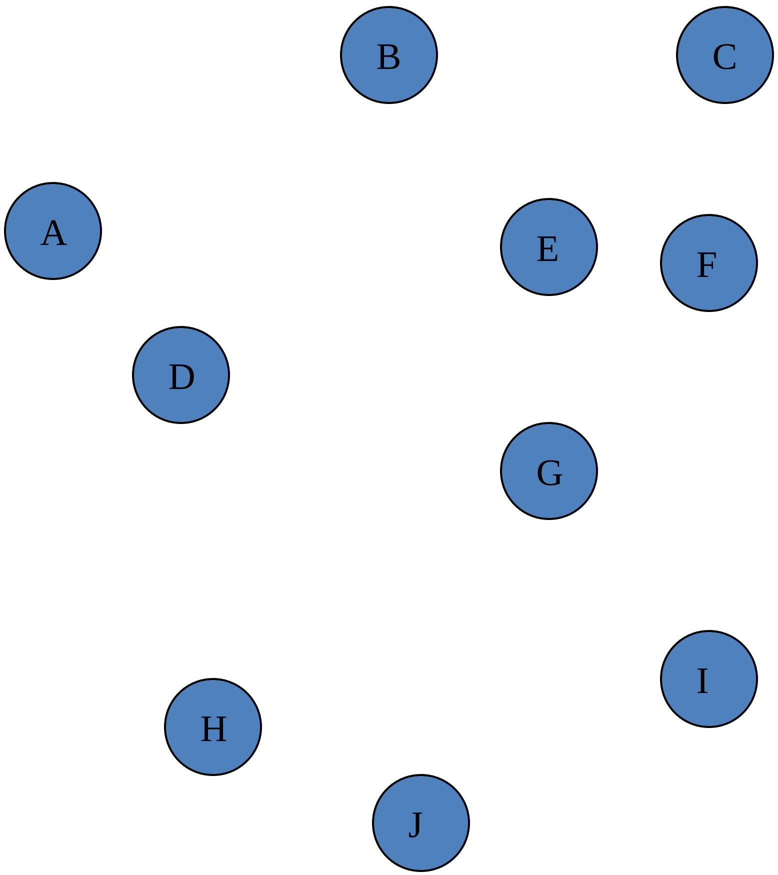
Objective:

Finding segmentation $S = (C_1, C_2, \dots, C_r)$

1. Sort E into $(o_1, o_2, \dots, o_m)$ by non-decreasing edge weights.
2. Start with S^0, where each vertex v_i is a component.
3. Repeat steps below for $q = 1, \dots, m$ to construct S^q given S^{q-1}:
    1. Let v_i in C_i^{q-1} and v_j in C_j^{q-1} denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$.
    2. If $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$ then merge two components.
4. *Return* $S = S^m$.

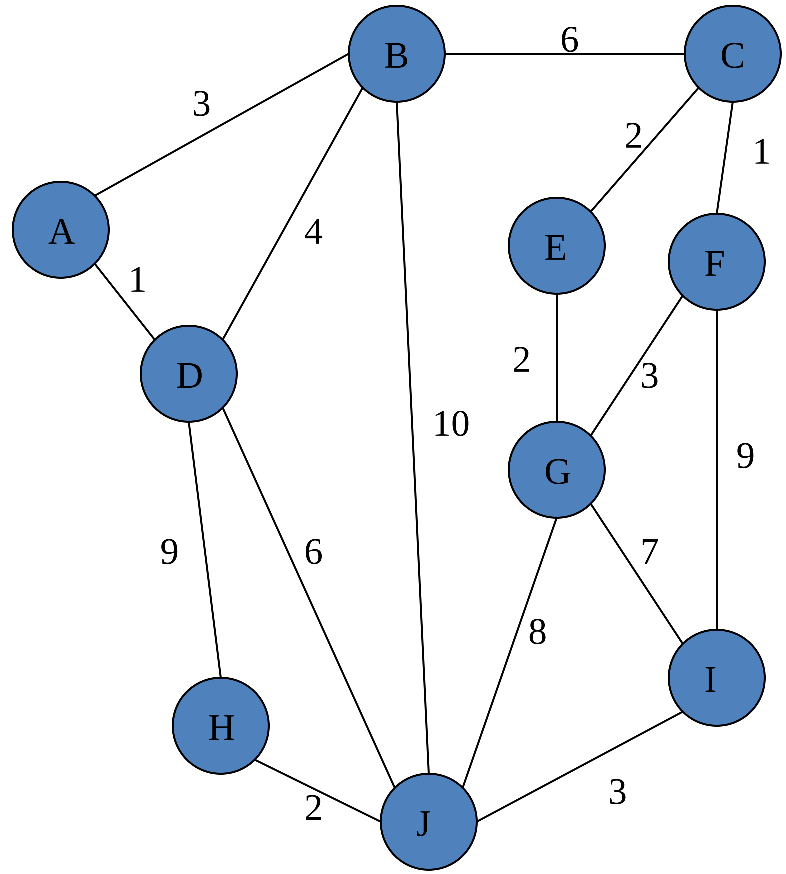
Minimum Spanning Tree

(Kruskal Algorithm)



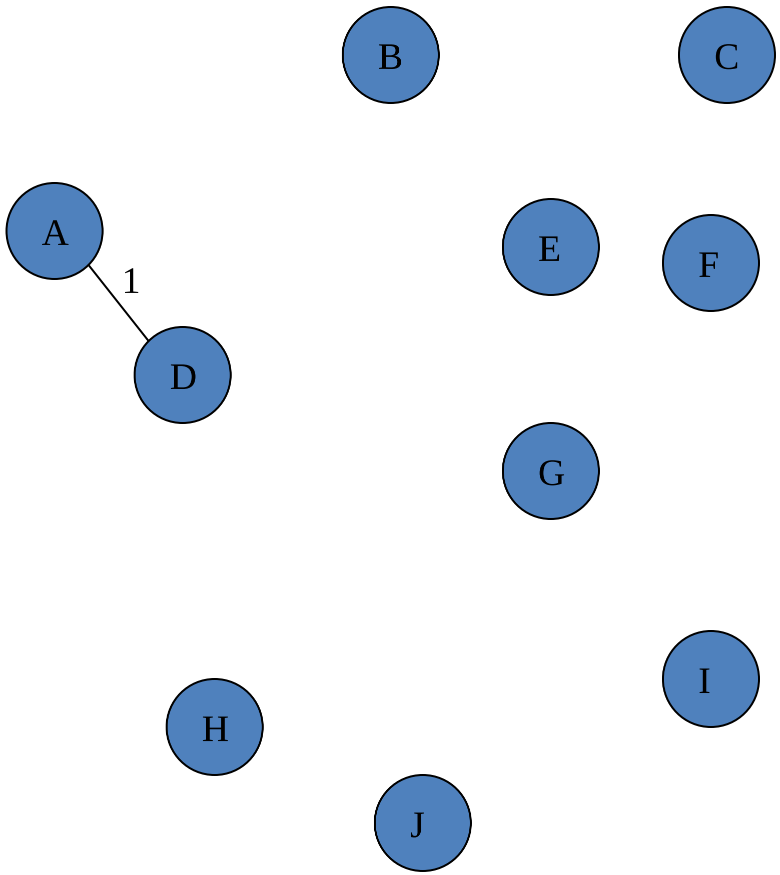
$\boxed{\tau = 2}$

Complete Graph

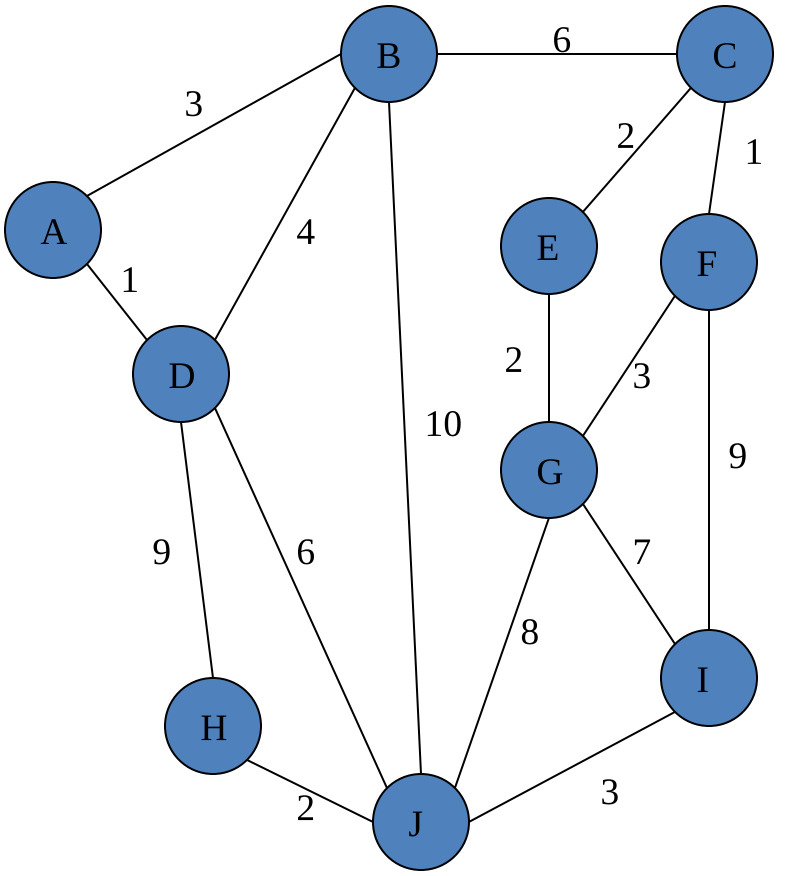


Minimum Spanning Tree

(Kruskal Algorithm)

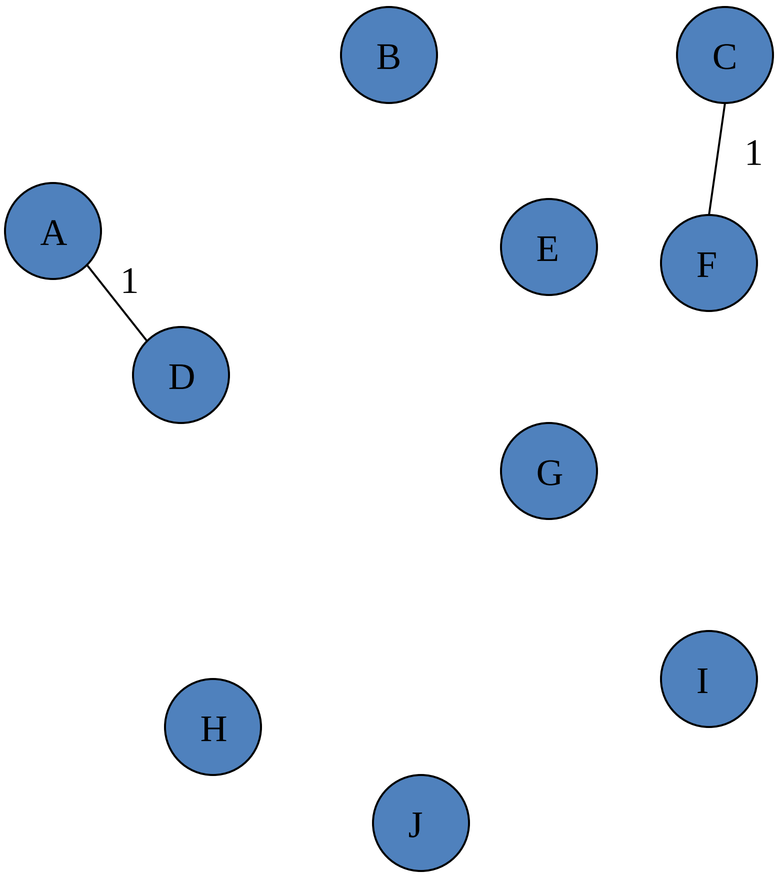


Complete Graph



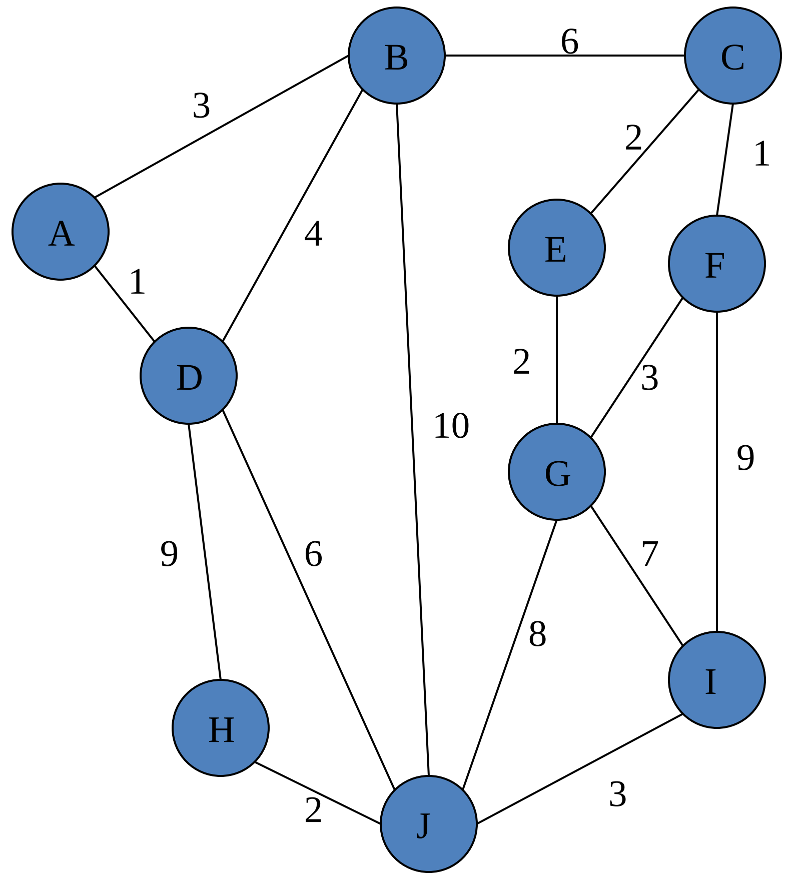
$\tau = 2$

Minimum Spanning Tree (Kruskal Algorithm)

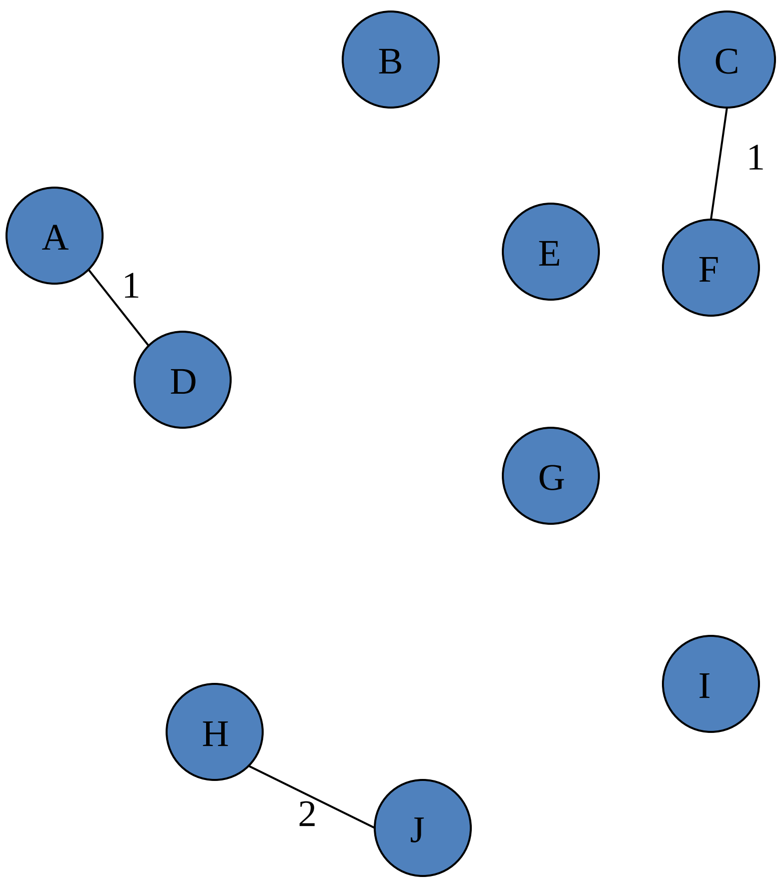


$\tau = 2$

Complete Graph

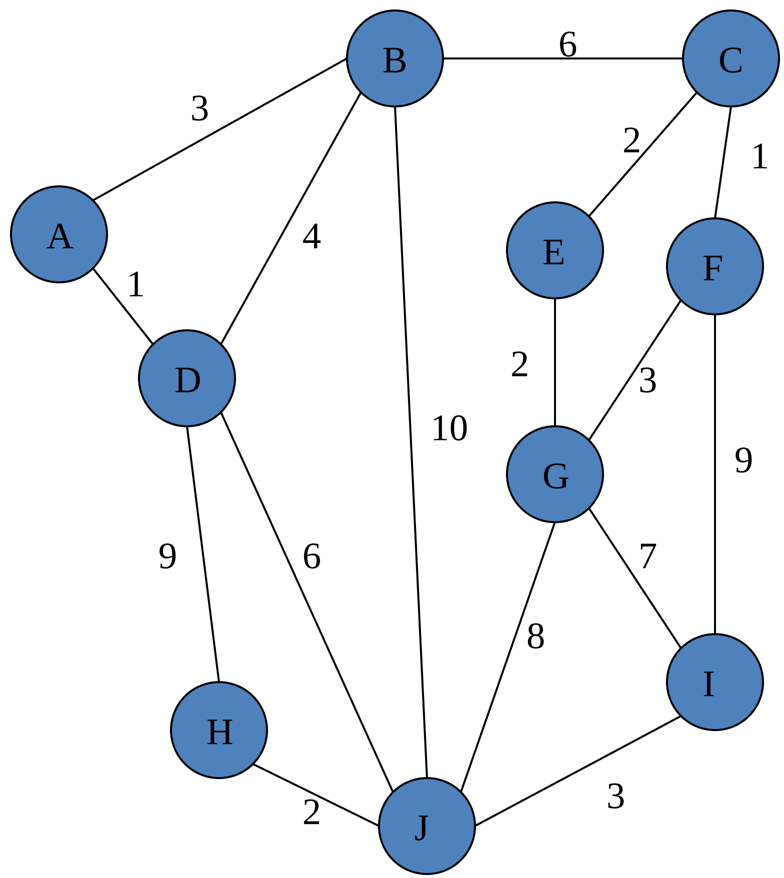


Minimum Spanning Tree (Kruskal Algorithm)



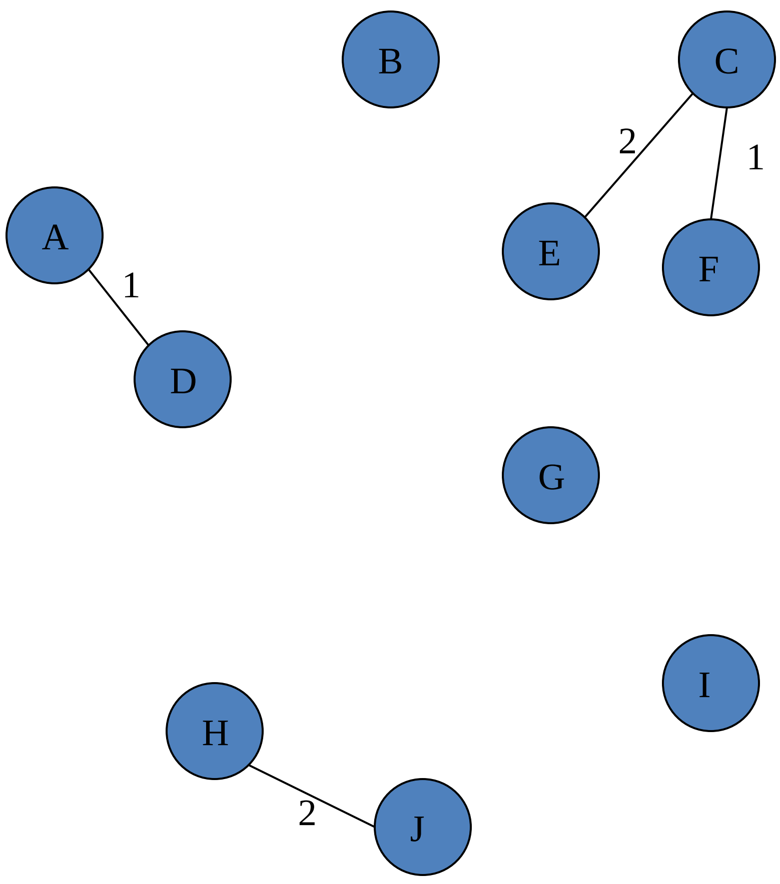
$\boxed{\tau = 2}$

Complete Graph



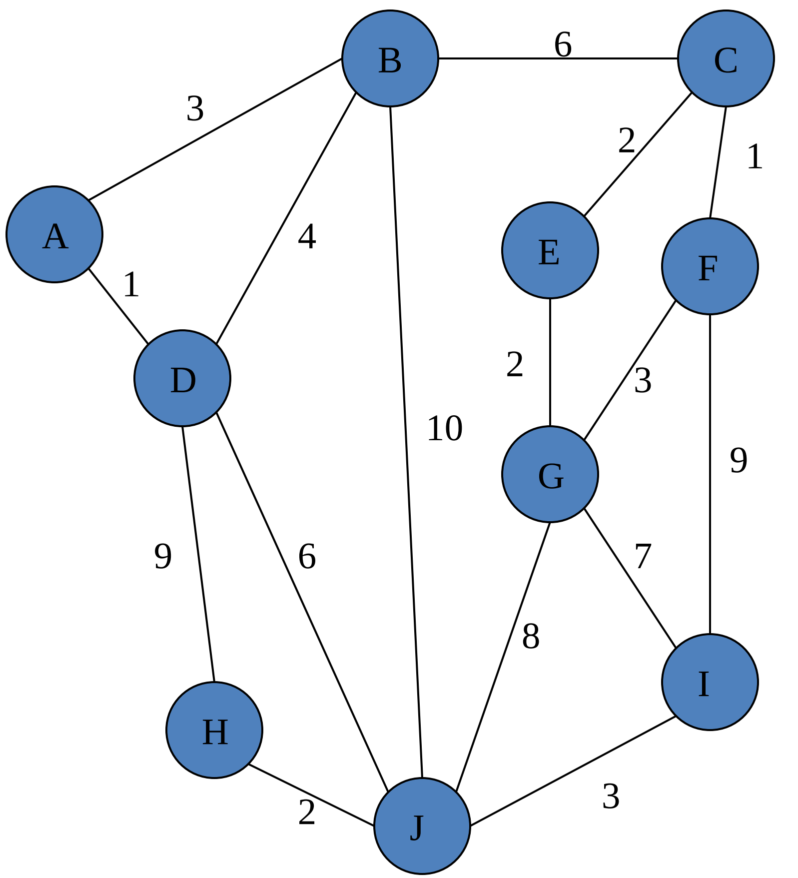
Minimum Spanning Tree

(Kruskal Algorithm)

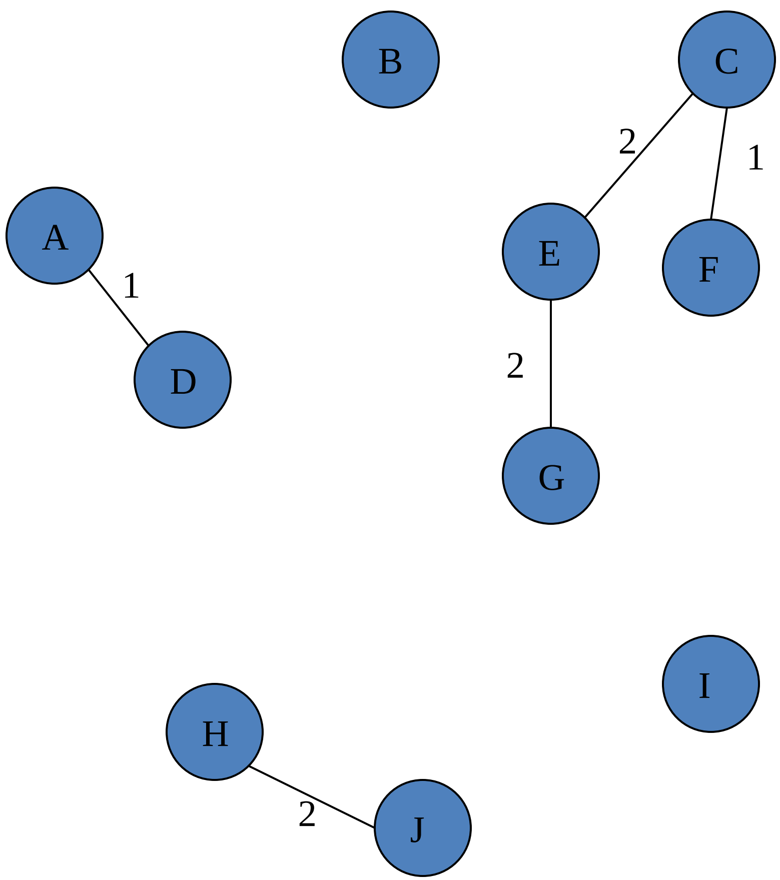


$\boxed{\tau = 2}$

Complete Graph

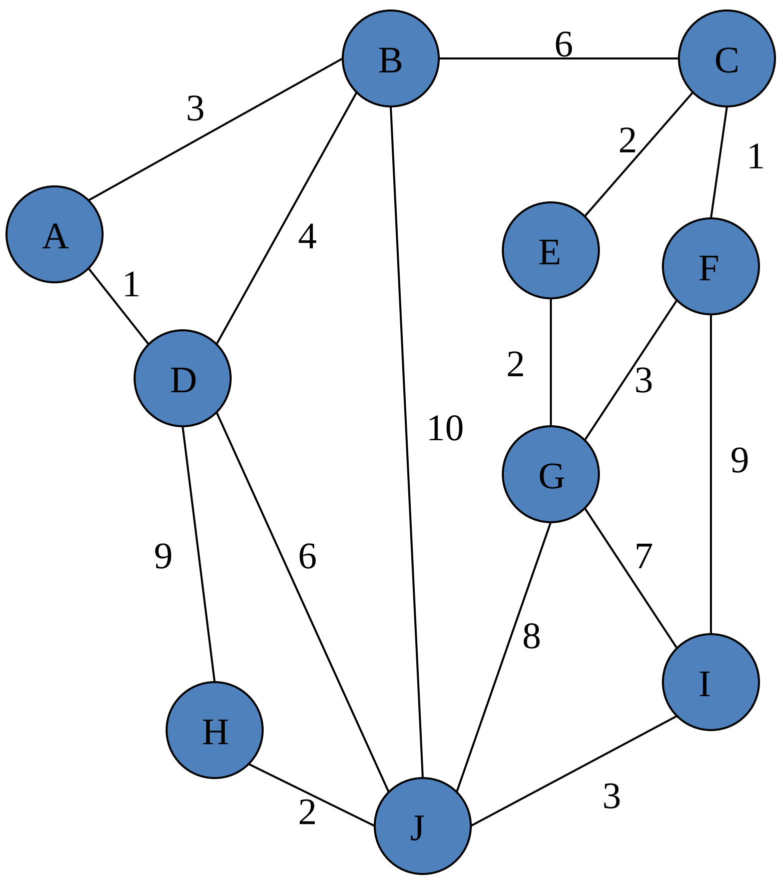


Minimum Spanning Tree (Kruskal Algorithm)



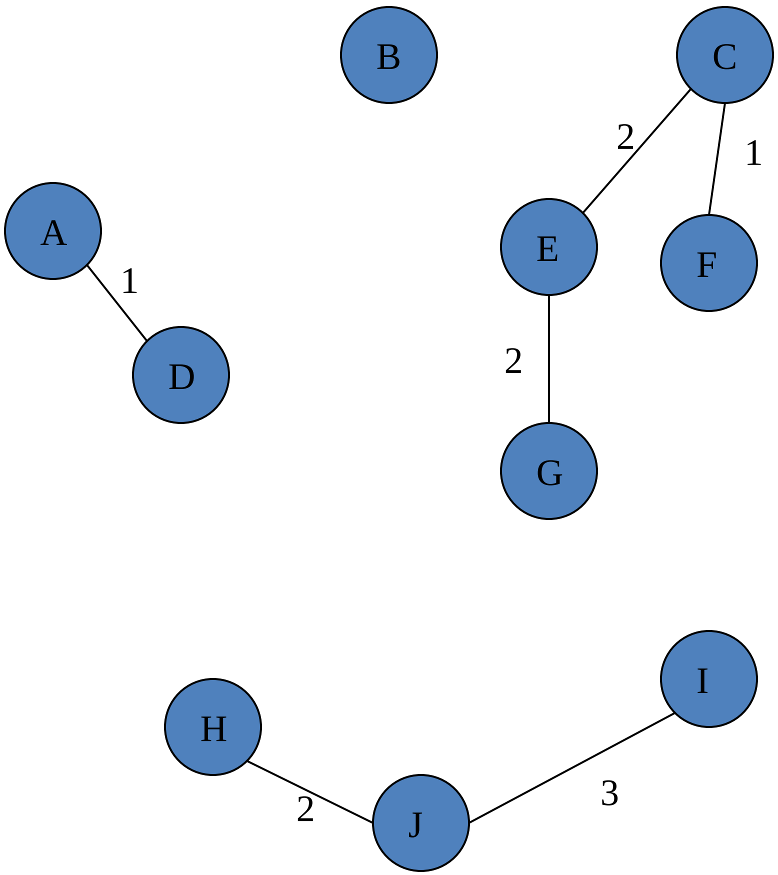
$\boxed{\tau = 2}$

Complete Graph



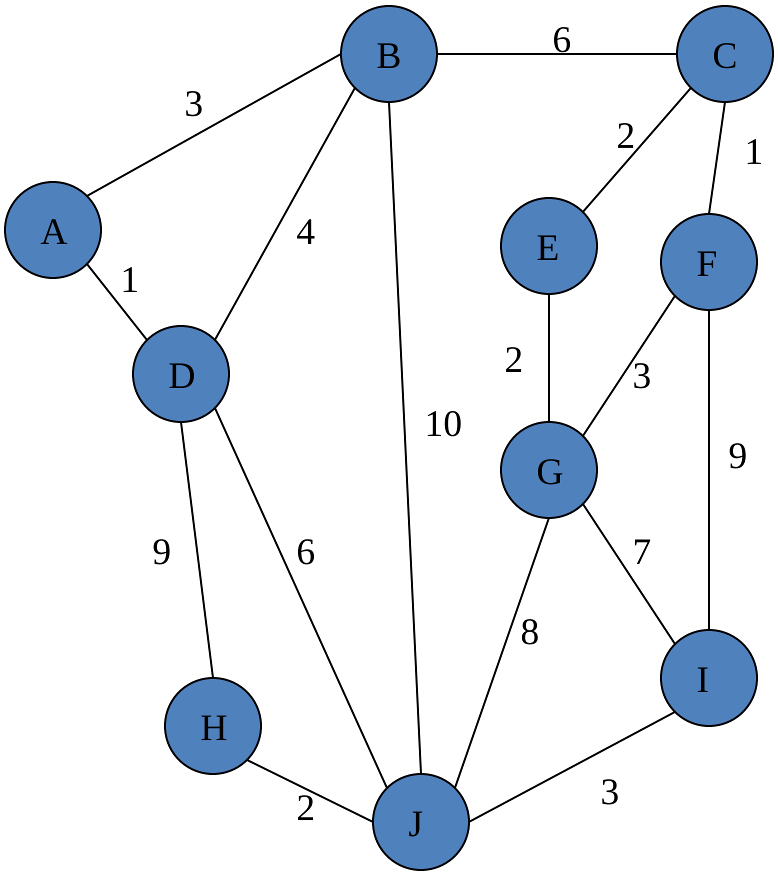
Minimum Spanning Tree

(Kruskal Algorithm)

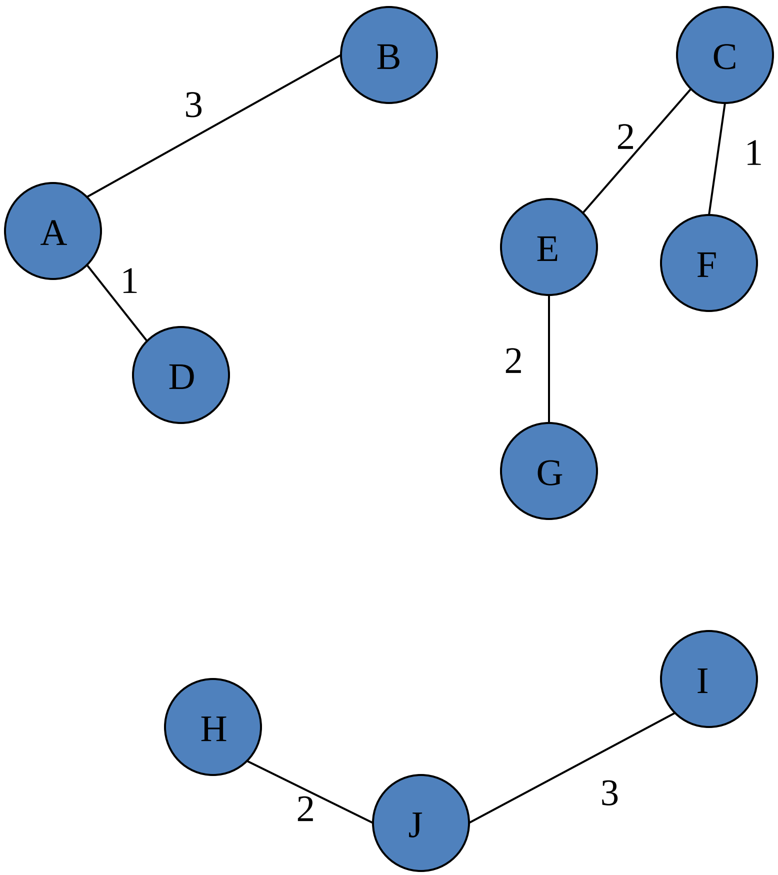


$\tau = 2$

Complete Graph

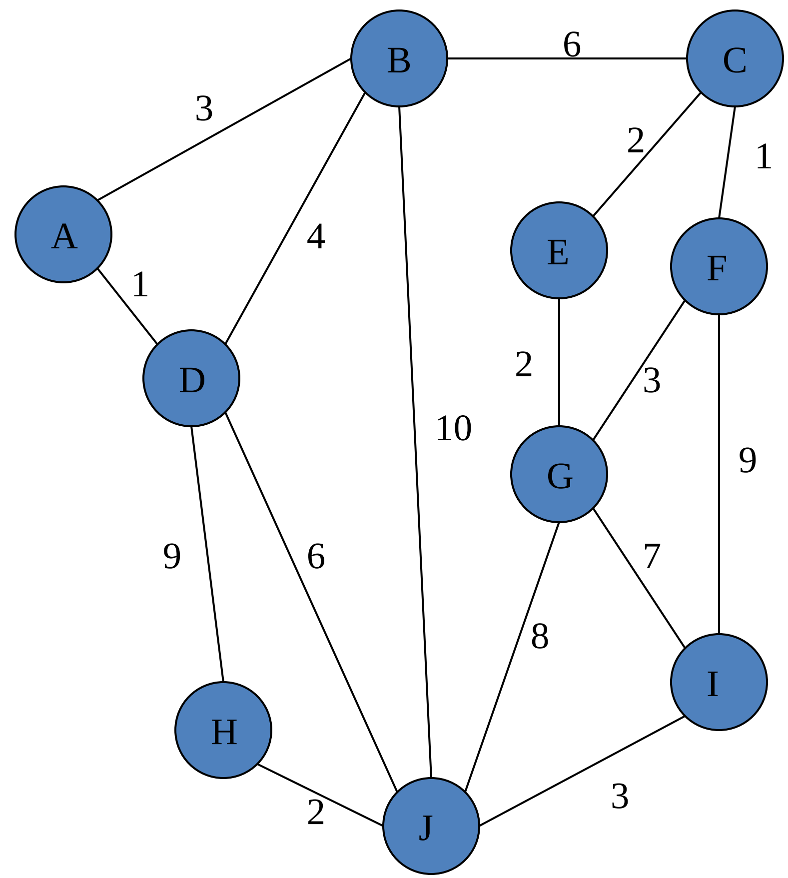


Minimum Spanning Tree
(Kruskal Algorithm)

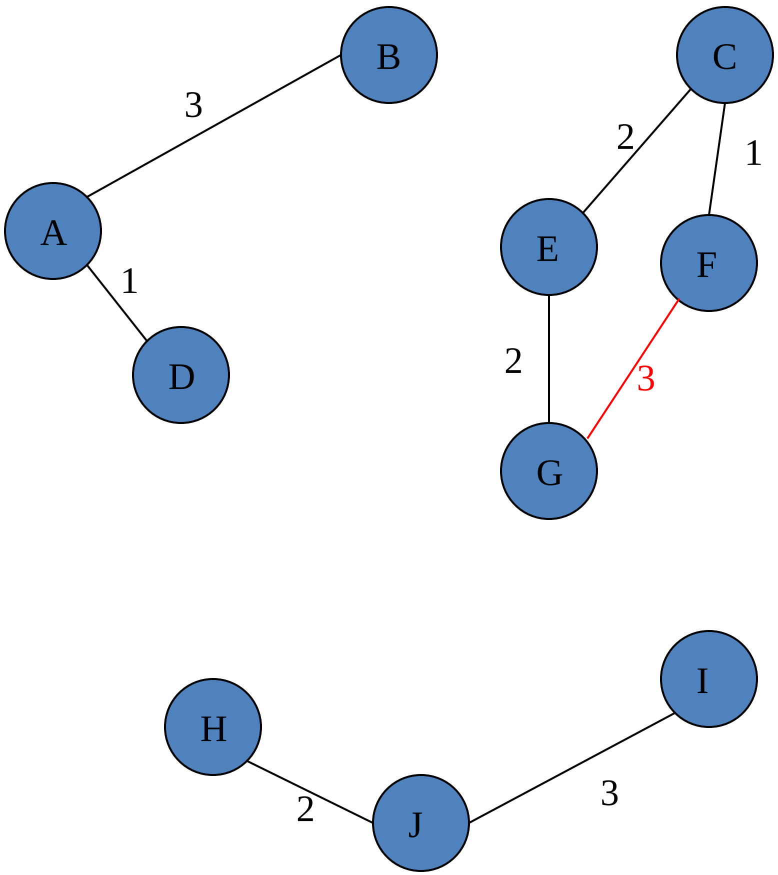


$\boxed{\tau = 2}$

Complete Graph

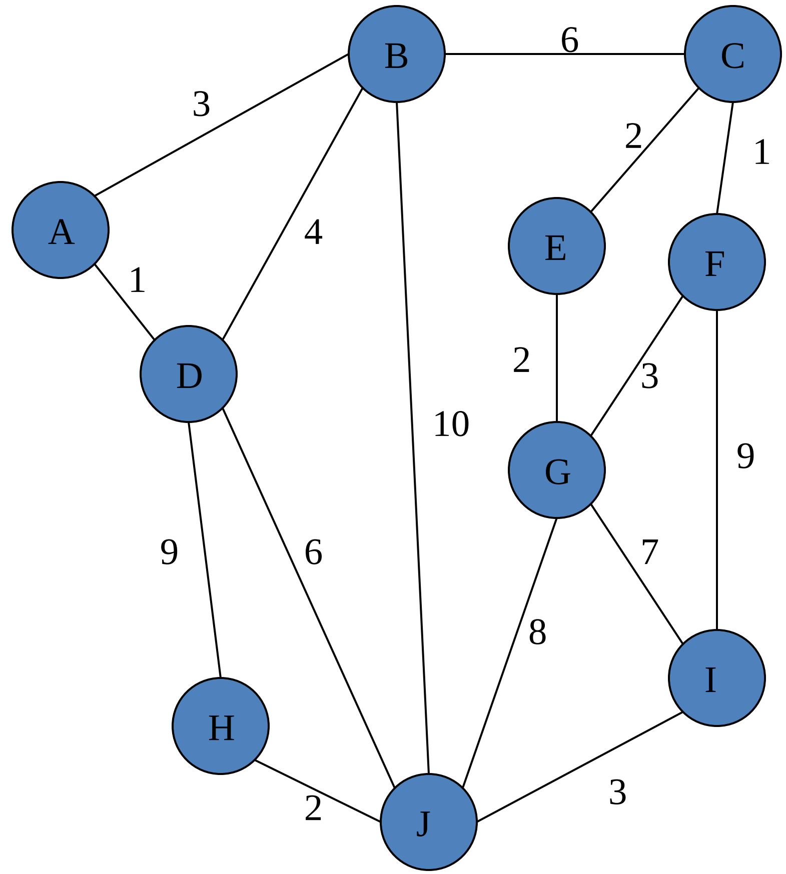


Minimum Spanning Tree (Kruskal Algorithm)



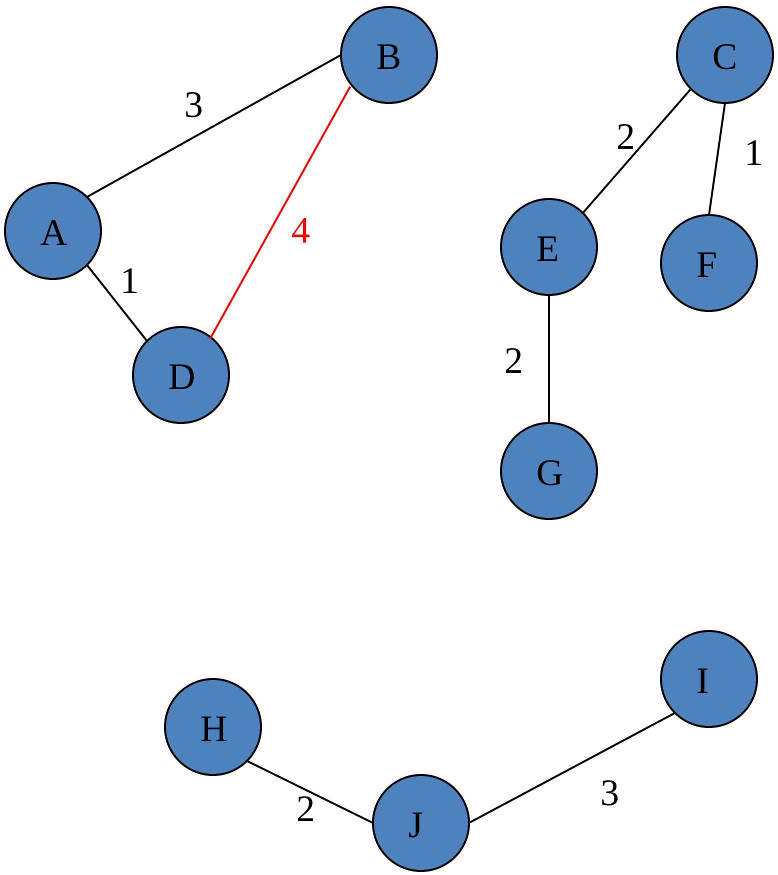
$\boxed{\tau = 2}$

Complete Graph



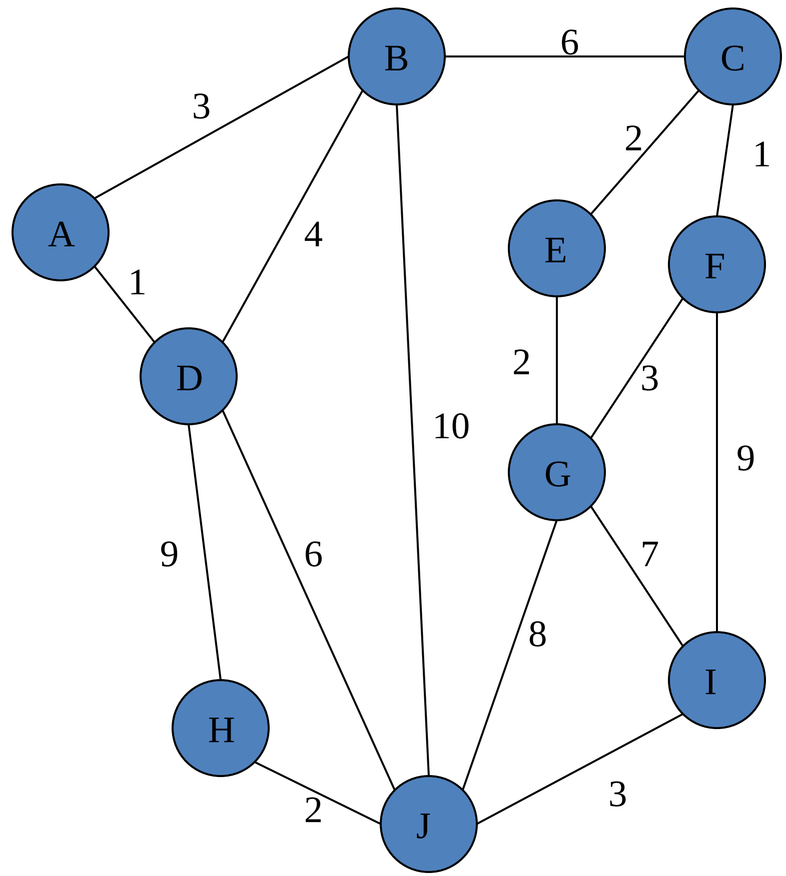
Minimum Spanning Tree

(Kruskal Algorithm)



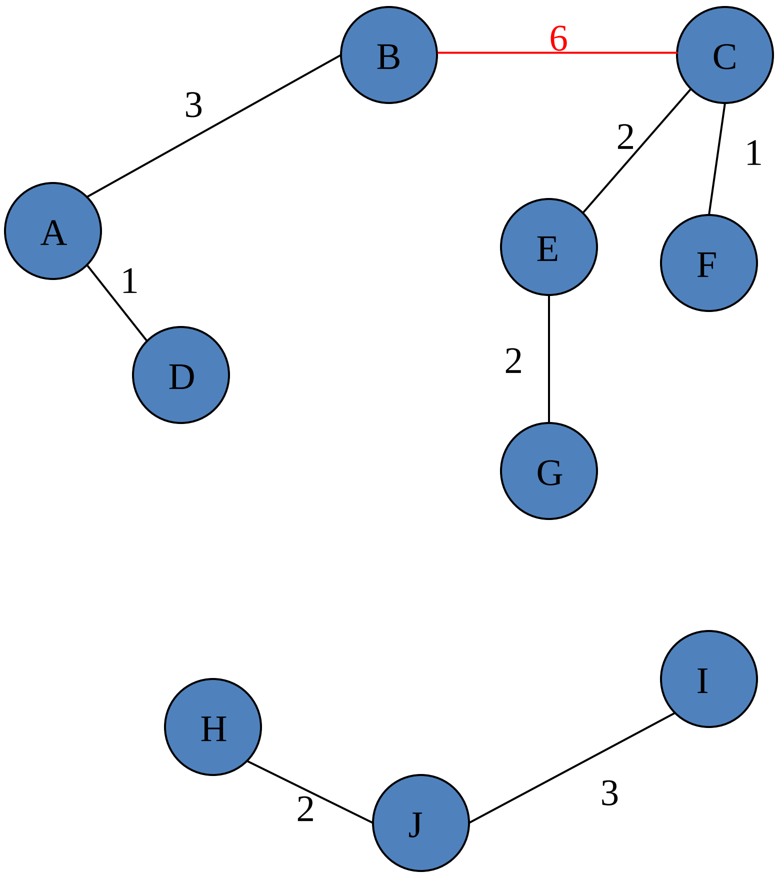
$\tau = 2$

Complete Graph



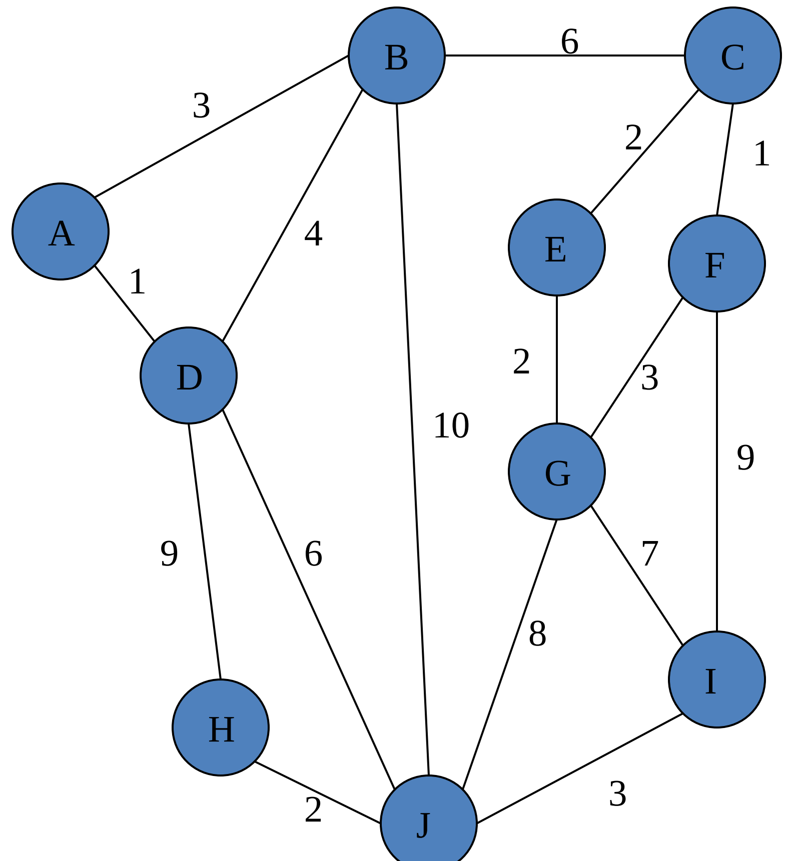
Minimum Spanning Tree

(Kruskal Algorithm)



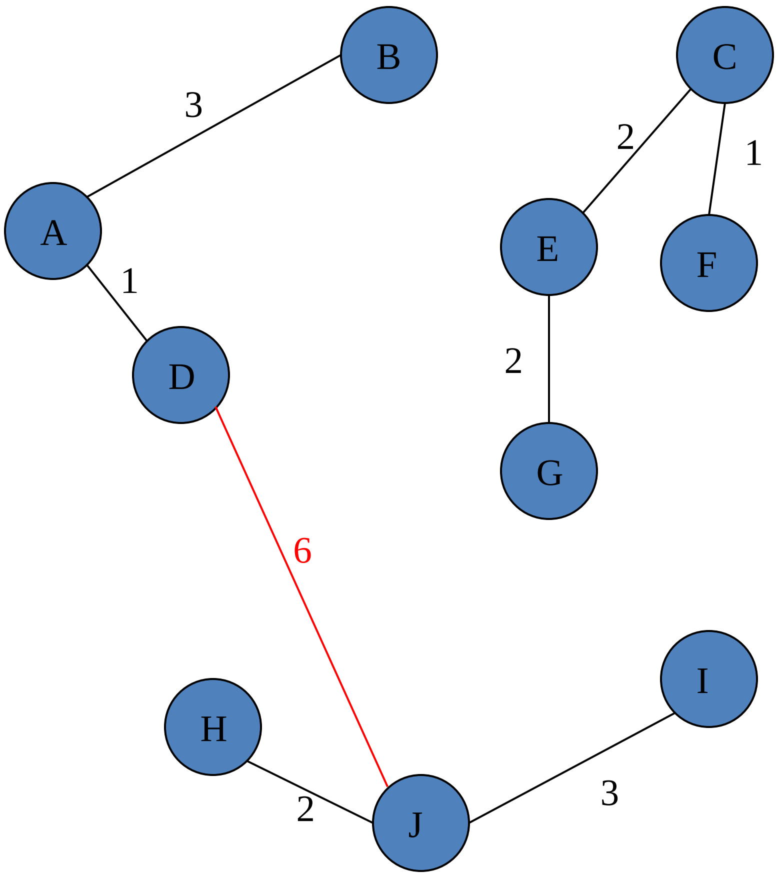
$\boxed{\tau = 2}$

Complete Graph



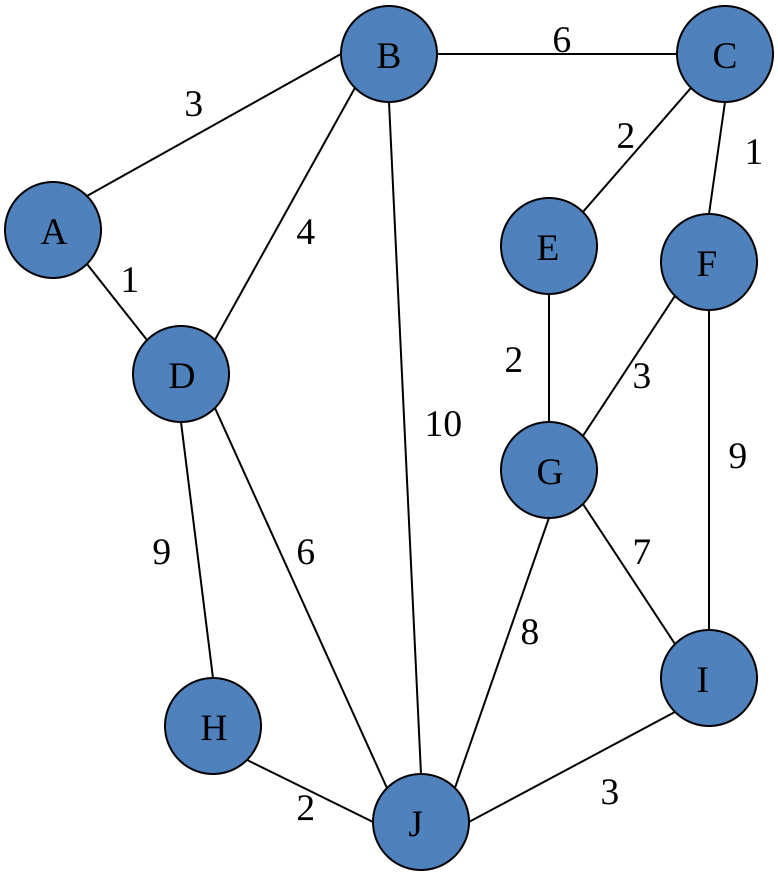
Minimum Spanning Tree

(Kruskal Algorithm)



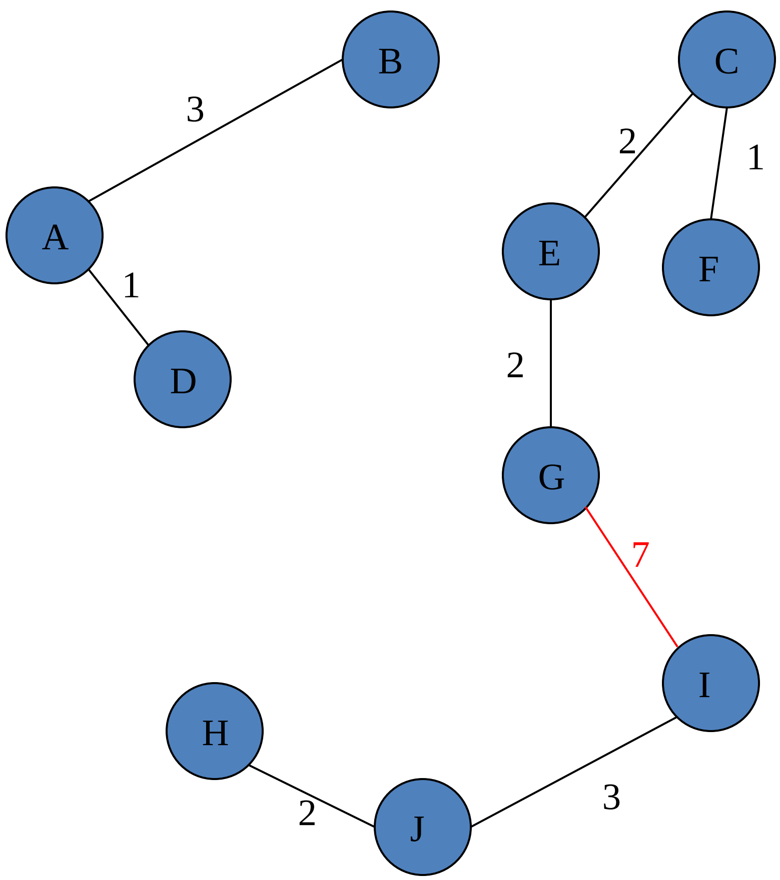
$\boxed{\tau = 2}$

Complete Graph



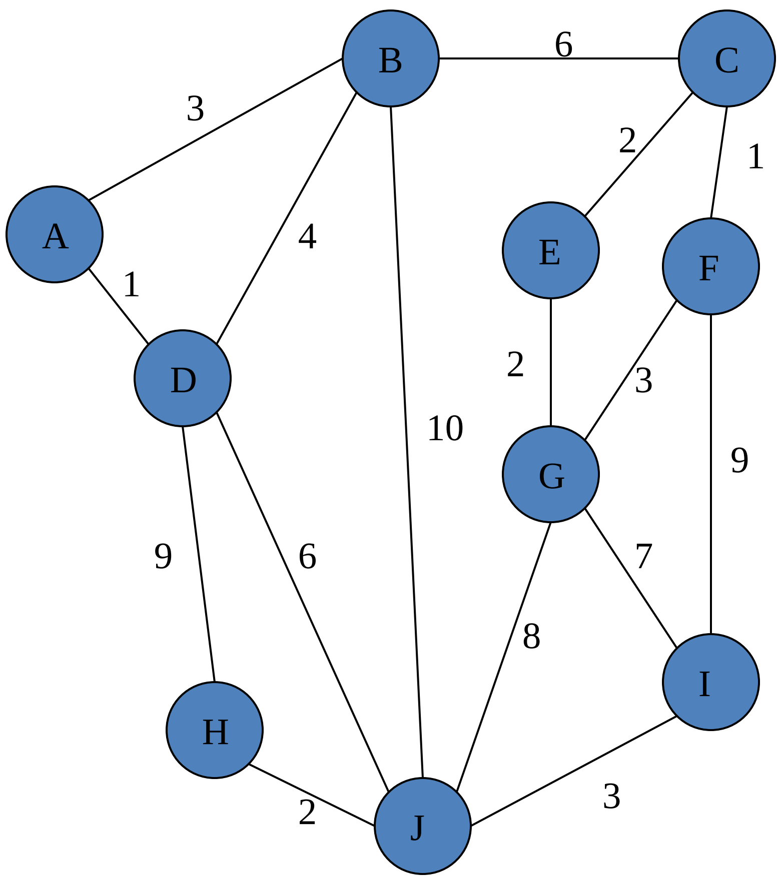
Minimum Spanning Tree

(Kruskal Algorithm)



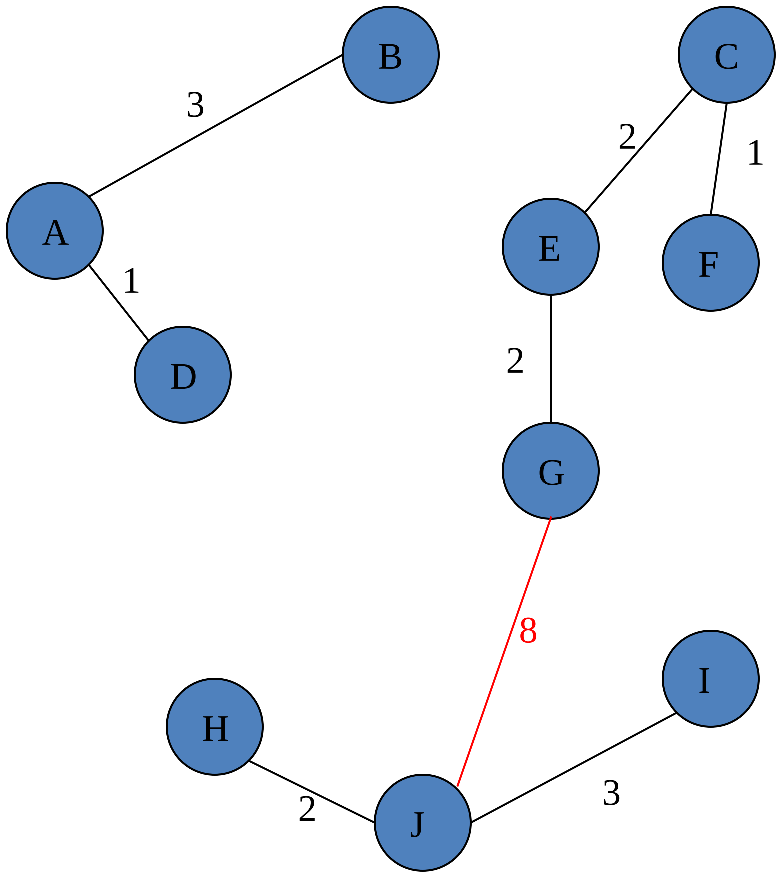
$\tau = 2$

Complete Graph



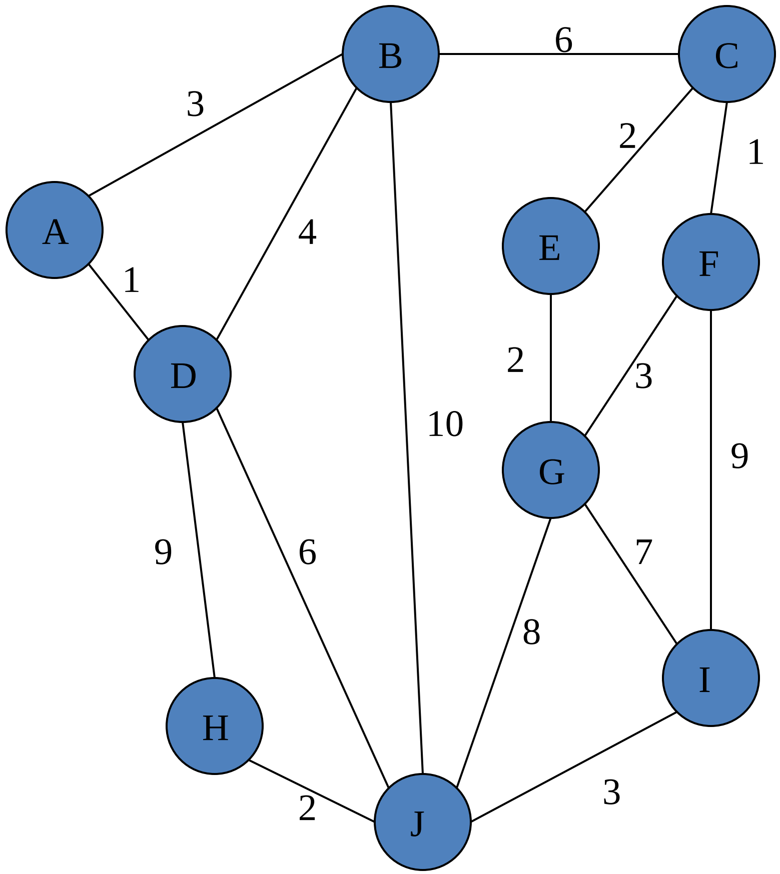
Minimum Spanning Tree

(Kruskal Algorithm)



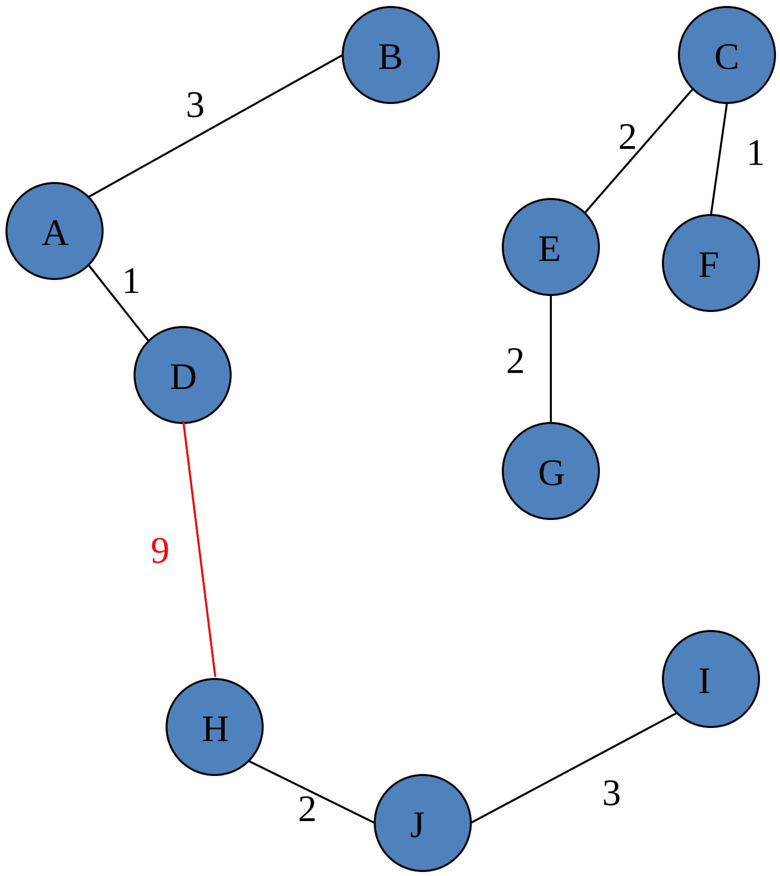
$\boxed{\tau = 2}$

Complete Graph



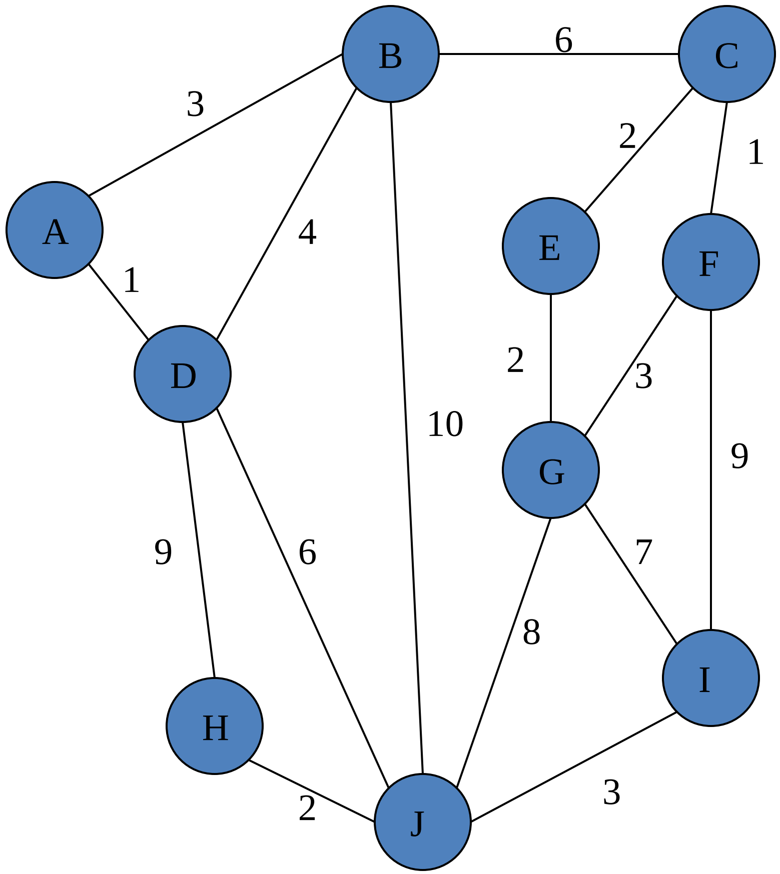
Minimum Spanning Tree

(Kruskal Algorithm)

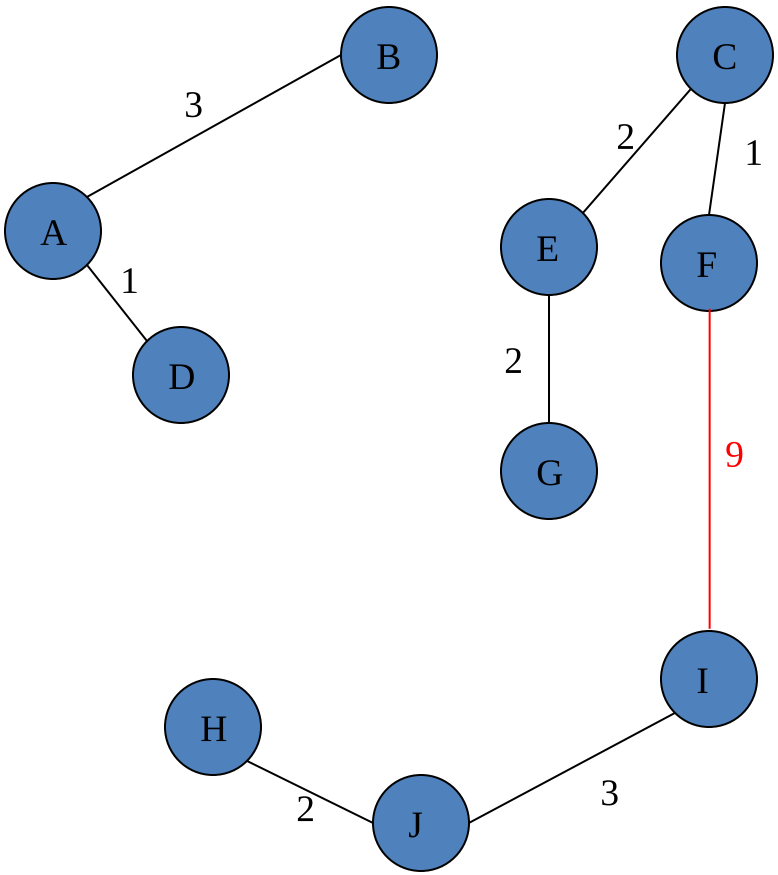


$\tau = 2$

Complete Graph

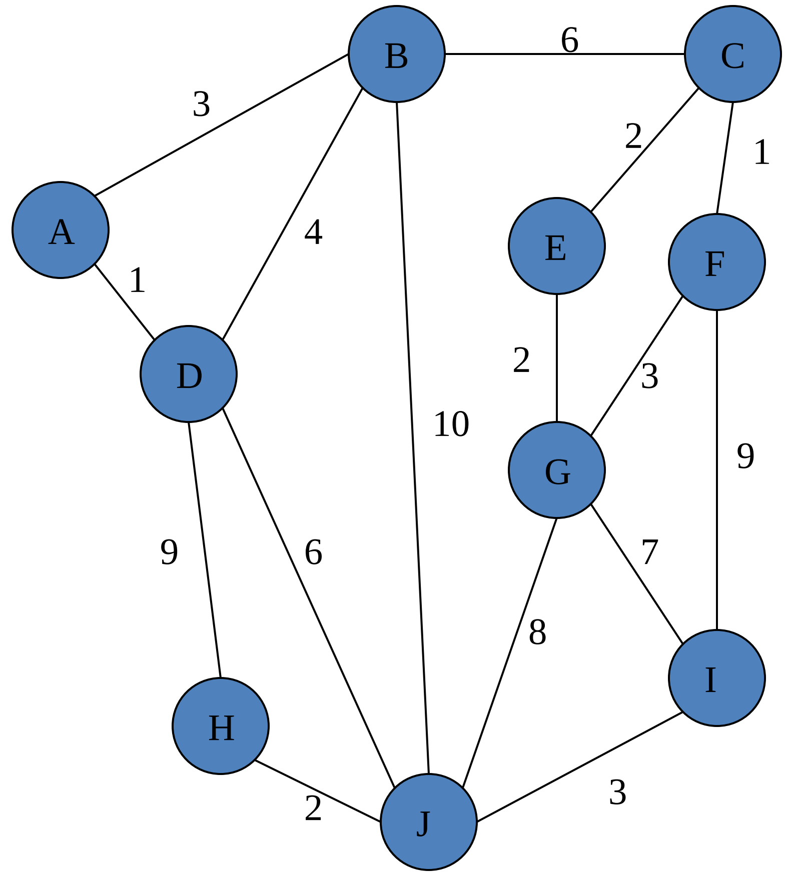


Minimum Spanning Tree (Kruskal Algorithm)



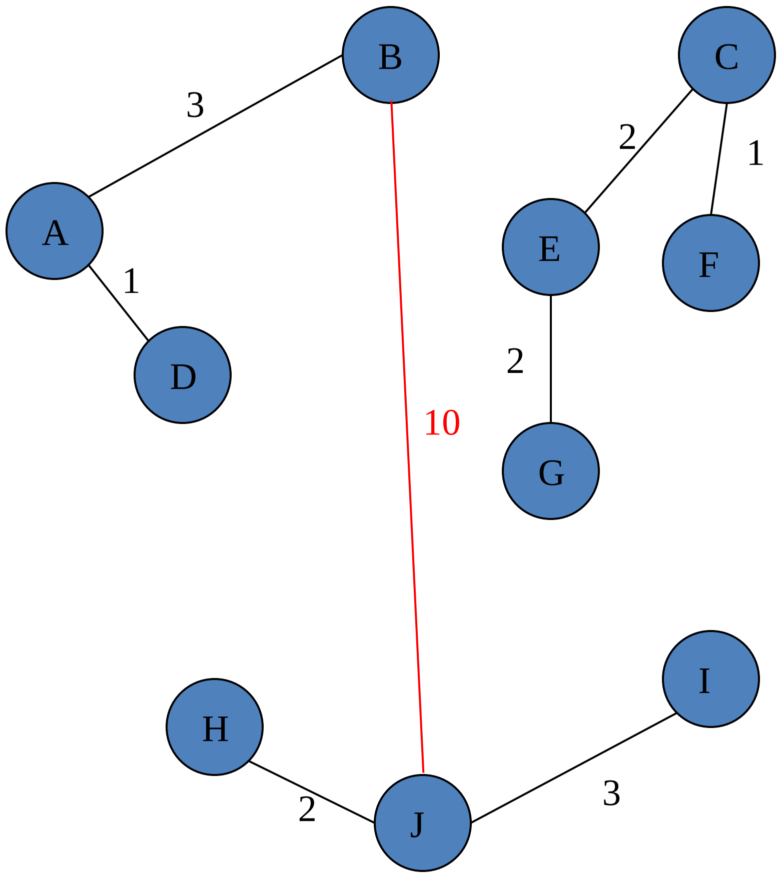
$\tau = 2$

Complete Graph



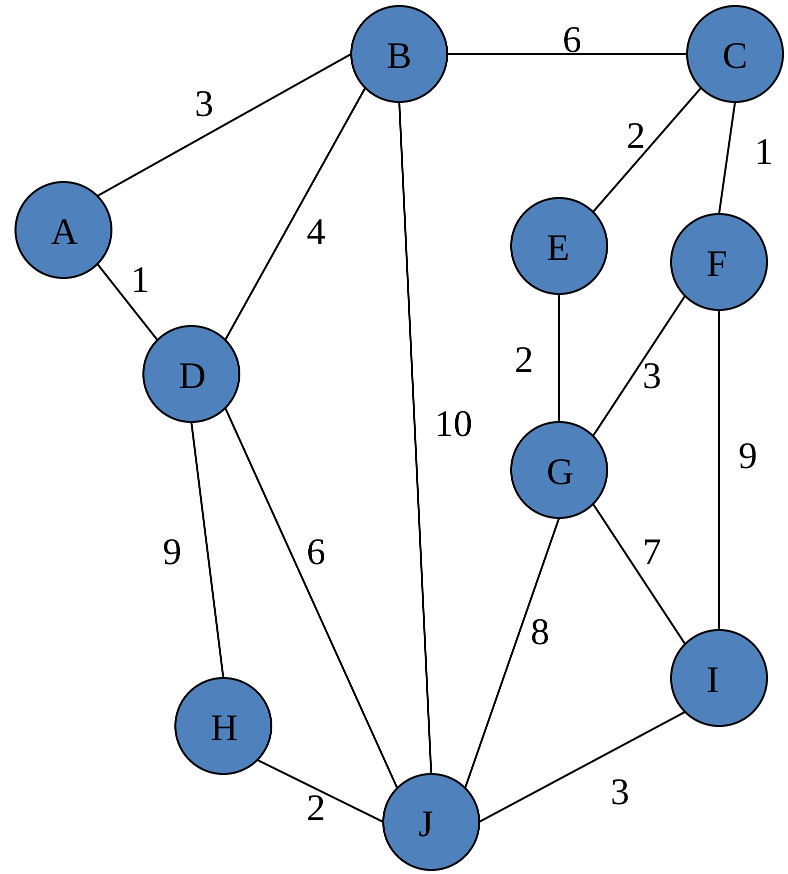
Minimum Spanning Tree

(Kruskal Algorithm)



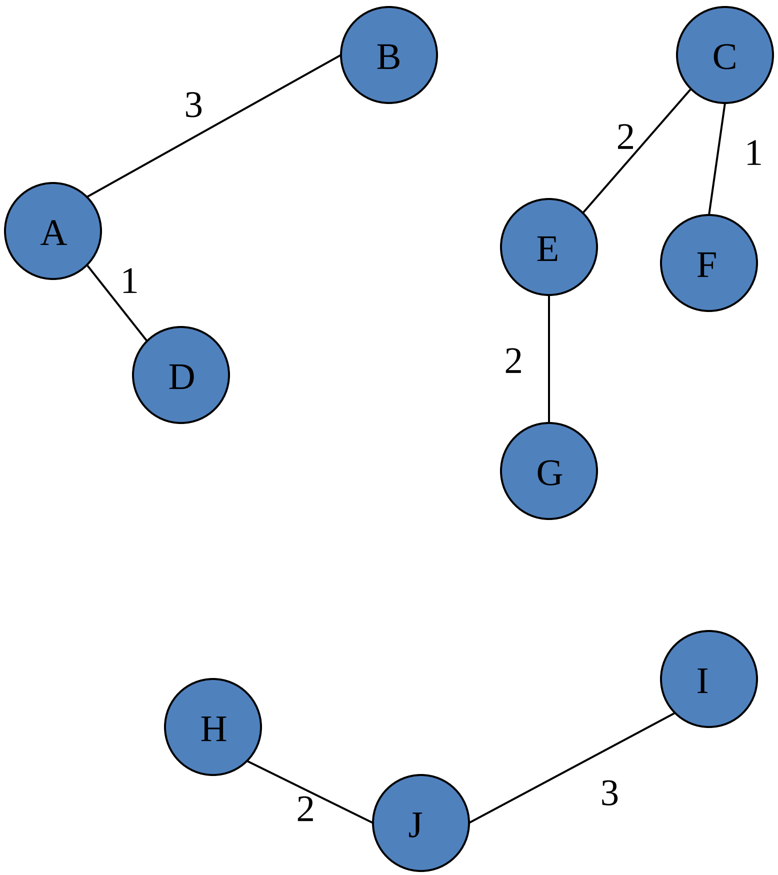
$\boxed{\tau = 2}$

Complete Graph



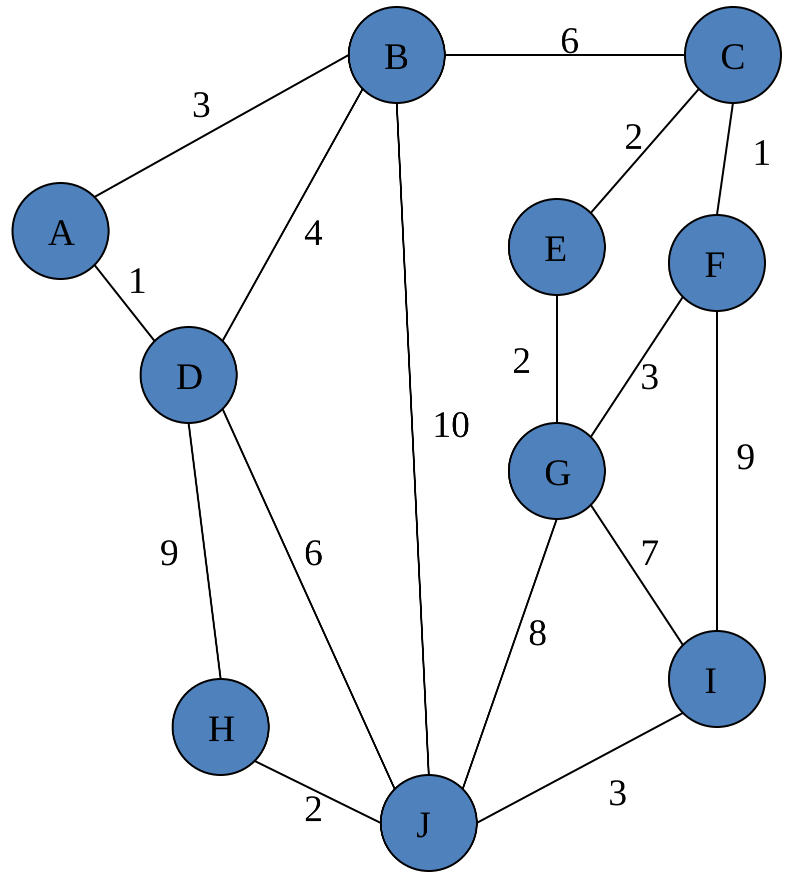
Minimum Spanning Tree

(Kruskal Algorithm)

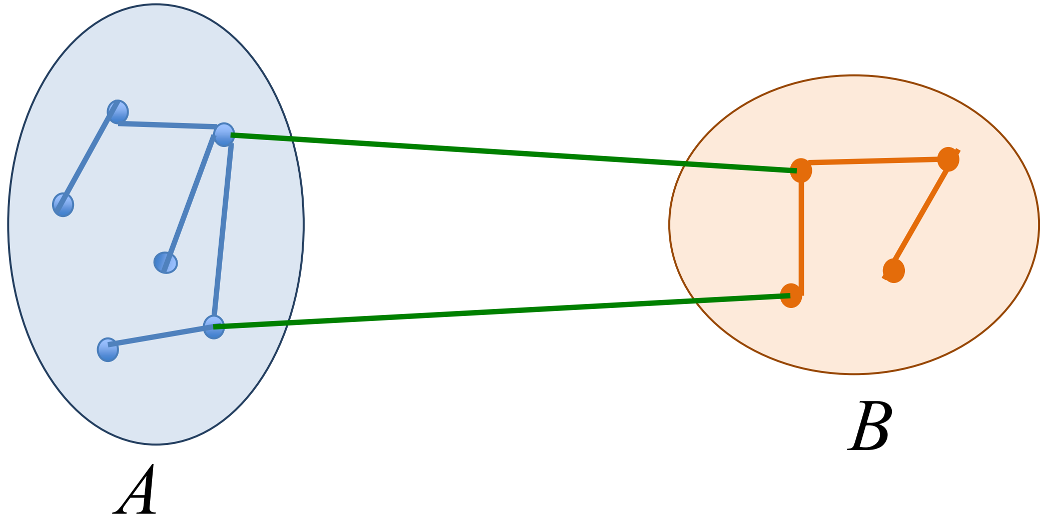


$\tau = 2$

Complete Graph



Optional



$$Int(A) = \max_{e \in MST(A,E)} w(e) \qquad Int(B) = \max_{e \in MST(A,E)} w(B)$$

$$MInt(A, B\ ) = \min(\ Int(A) + \tau(A)\ ,\ Int(B) + \tau(B)\ )$$

$$Dif(A, B) = \min_{\substack{v_i \in A,\ v_j \in B \\ (v_i, v_j) \in E}} w\big((v_i, v_j)\big)$$

$$\begin{cases} if\ Dif(A,B) < MInt(A,B) & Merge \\ if\ Dif(A,B) \geq MInt(A,B) & Don't\ merge \end{cases}$$

Refinement: Segmentation T is a refinement of segmentation S when each component of T is contained in S.

Too Fine: A segmentation is *too fine* if there is some pair of regions for which there is no evidence for a boundary between them.

Too Coarse: A segmentation is *too coarse* when there exists a proper refinement that is not too fine.

For any graph, there exists a segmentation that is neither too fine nor too coarse.

Lemma: In the algorithm, if two distinct components are considered and not merged, then one of these components will be in the final segmentation.

Optional

Theorem: The segmentation produced by the algorithm is neither too coarse nor too fine.

Theorem: The segmentation produced by the algorithm does not depend on which non-decreasing weight order of the edge is used.

Efficient Graph-Based Image Segmentation (Felzenswalb & Huttenlocher)

http://www.cs.brown.edu/~pff/segment/

+ Good for thin regions
+ Fast
+ Easy to control coarseness of segmentations
+ Can include both large and small regions

 - Often creates regions with strange shapes
 - Sometimes makes very large errors