Kruskal’s Algorithm

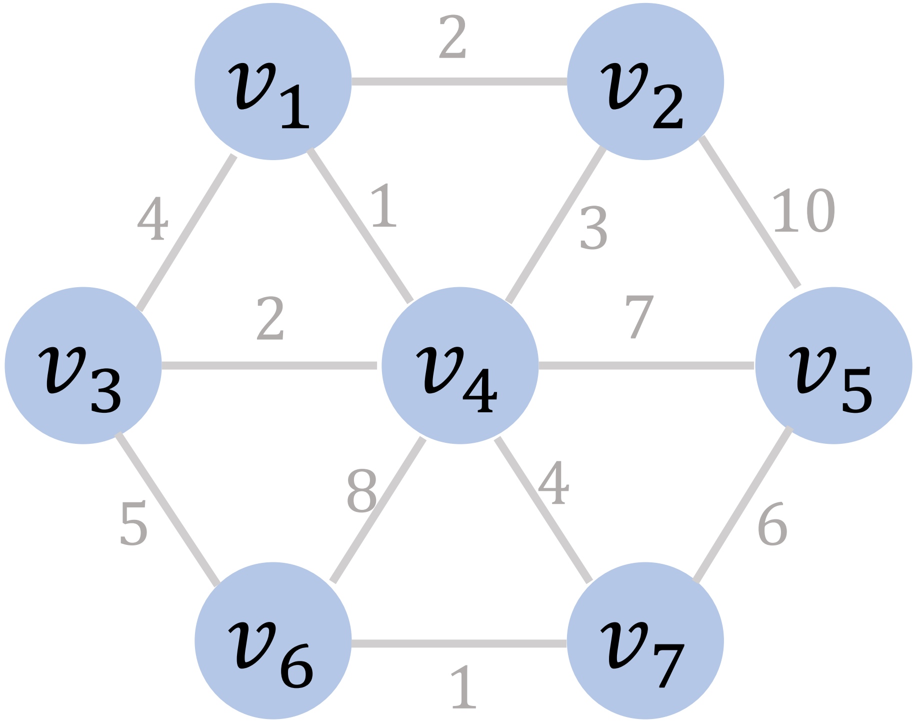
Shusen Wang

Kruskal's Algorithm

Basic idea: Maintain a forest, i.e., a collection of trees.

- Initially, there are n trees; every vertex is a tree.
- Each iteration studies one edge; the edge may be chosen so that two trees are merged.
- Stop when there is only one tree.
- The algorithm runs in at most $|\mathcal{E}|$ iterations. (Because there are $|\mathcal{E}|$ edges.)

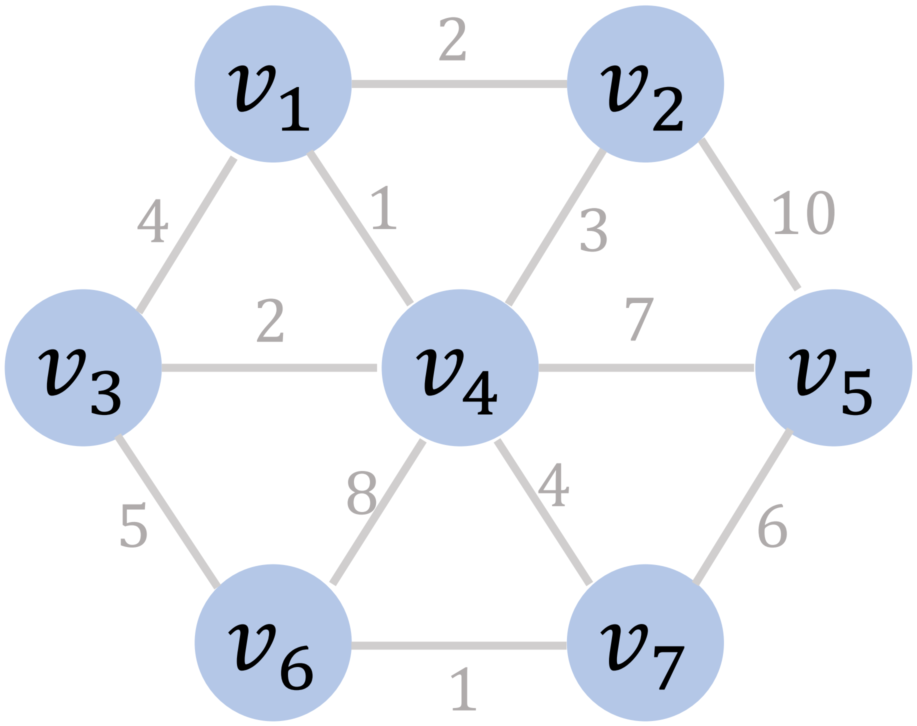
Preparations



- Build a queue of edges.
- Sort the elements so that the weights are in the ascending order.

Edge	Weight
$(1,4)$	1
$(6,7)$	1
$(1,2)$	2
$(3,4)$	2
$(2,4)$	3
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

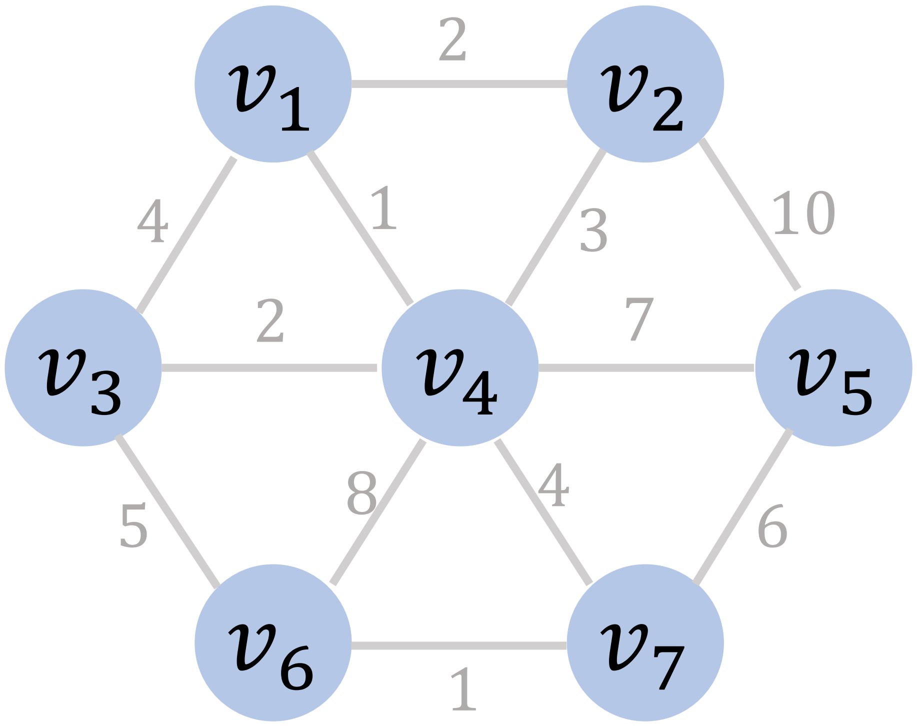
Preparations



- Build a queue of edges.
- Sort the elements so that the weights are in the ascending order.

Edge	Weight
$(1,4)$	1
$(6,7)$	1
$(1,2)$	2
$(3,4)$	2
$(2,4)$	3
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Preparations

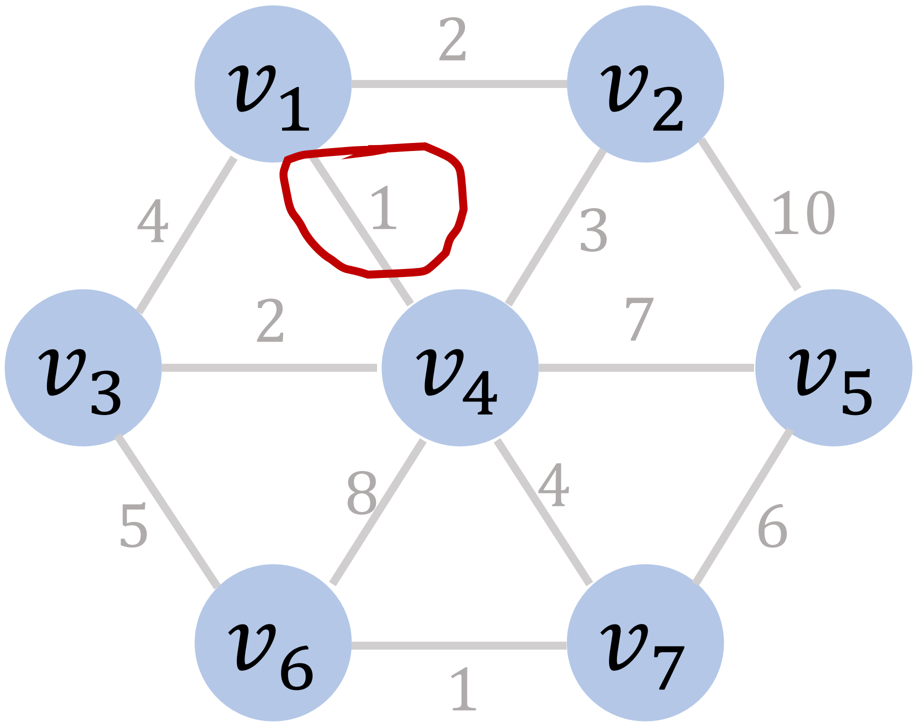


- Build a queue of edges.
- Sort the elements so that the weights are in the ascending order.

$\mathcal{T} = \emptyset$. (Record the selected edges.)

Edge	Weight
$(1,4)$	1
$(6,7)$	1
$(1,2)$	2
$(3,4)$	2
$(2,4)$	3
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Iteration 1

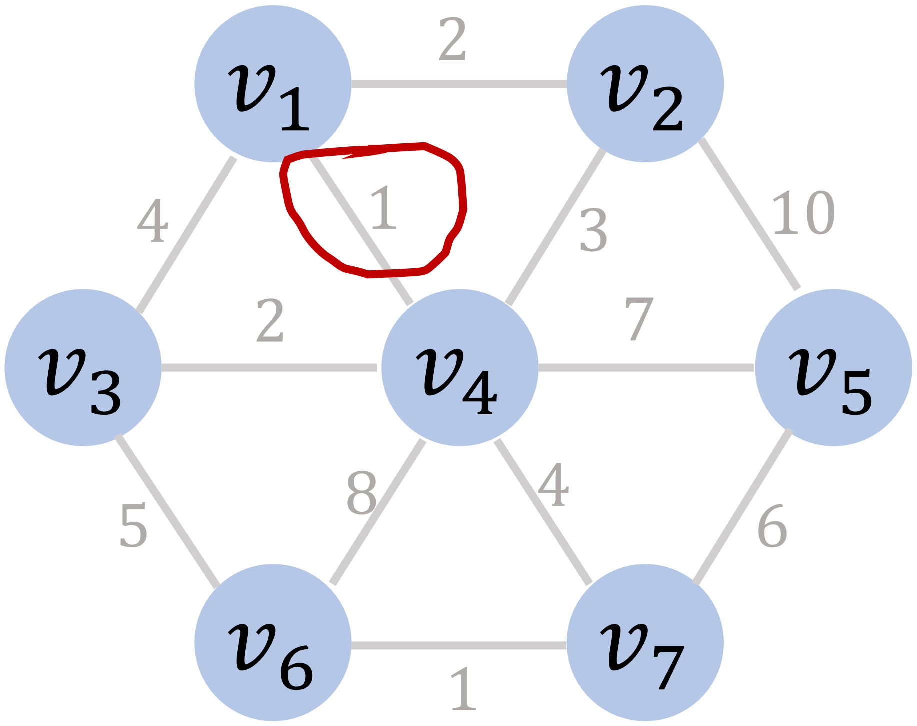


$\mathcal{T} = \emptyset$

- Perform dequeue and get the edge $(1, 4)$.

Edge	Weight
$(1, 4)$	1
$(6, 7)$	1
$(1, 2)$	2
$(3, 4)$	2
$(2, 4)$	3
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 1

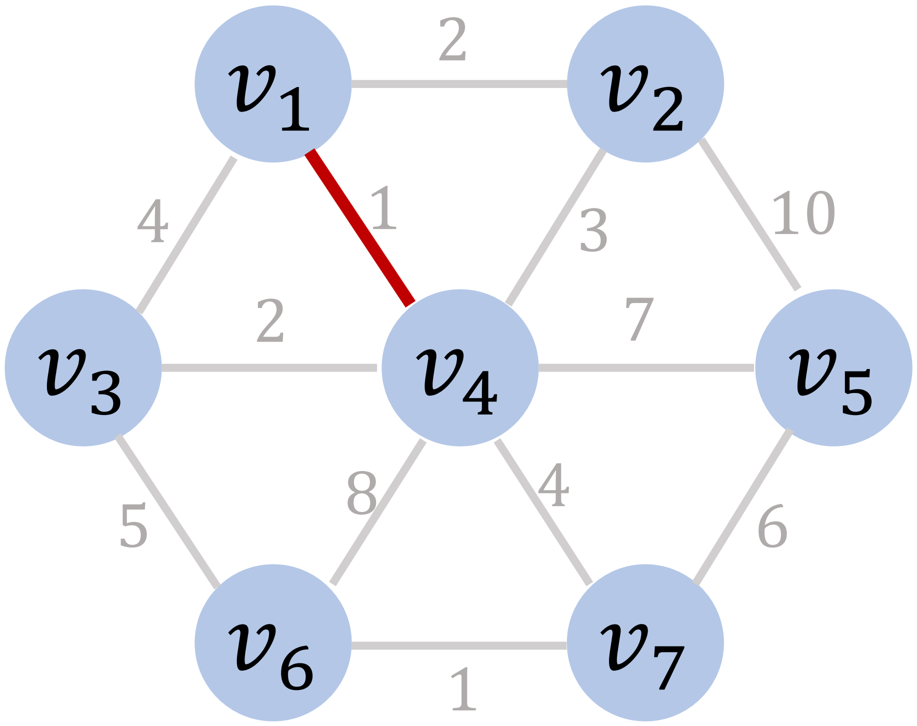


$\mathcal{T} = \emptyset$

- Perform dequeue and get the edge $(1,4)$.
- v_1 and v_4 are not in the same tree.
- Thus accept edge $(1,4)$.

Edge	Weight
$(6,7)$	1
$(1,2)$	2
$(3,4)$	2
$(2,4)$	3
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Iteration 1

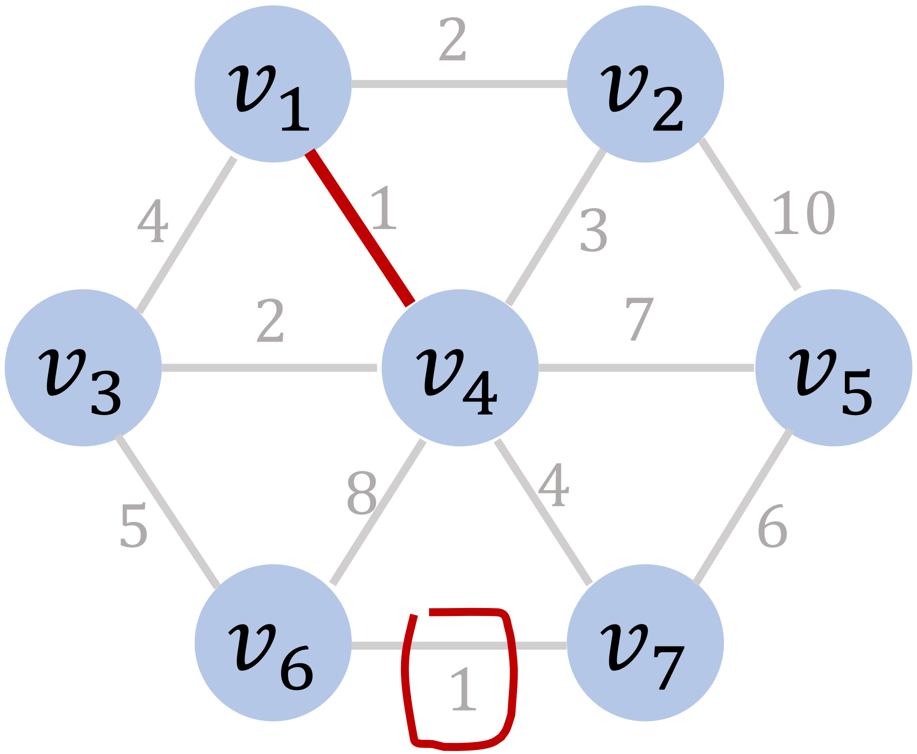


$\mathcal{T} = \{e_{1,4}\}$

- Perform dequeue and get the edge $(1,4)$.
- v_1 and v_4 are not in the same tree.
- Thus accept edge $(1,4)$.
- Append $(1,4)$ to $\mathcal{T}$.

Edge	Weight
$(6,7)$	1
$(1,2)$	2
$(3,4)$	2
$(2,4)$	3
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Iteration 2

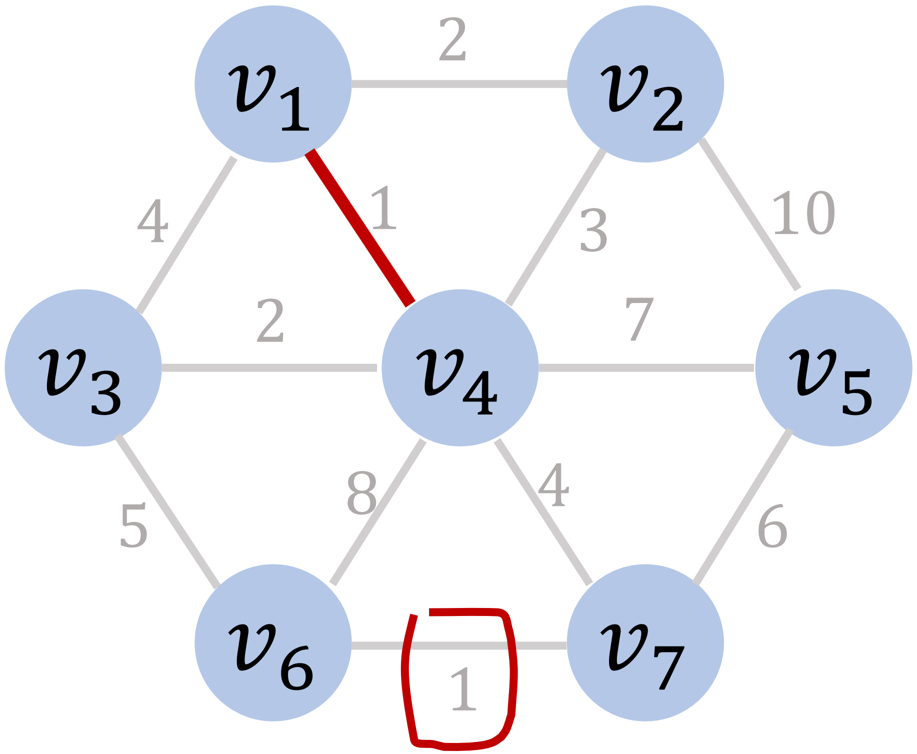


$\mathcal{T} = \{e_{1,4}\}$

- Perform dequeue and get the edge $(6,7)$.

Edge	Weight
$(6,7)$	1
$(1,2)$	2
$(3,4)$	2
$(2,4)$	3
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Iteration 2

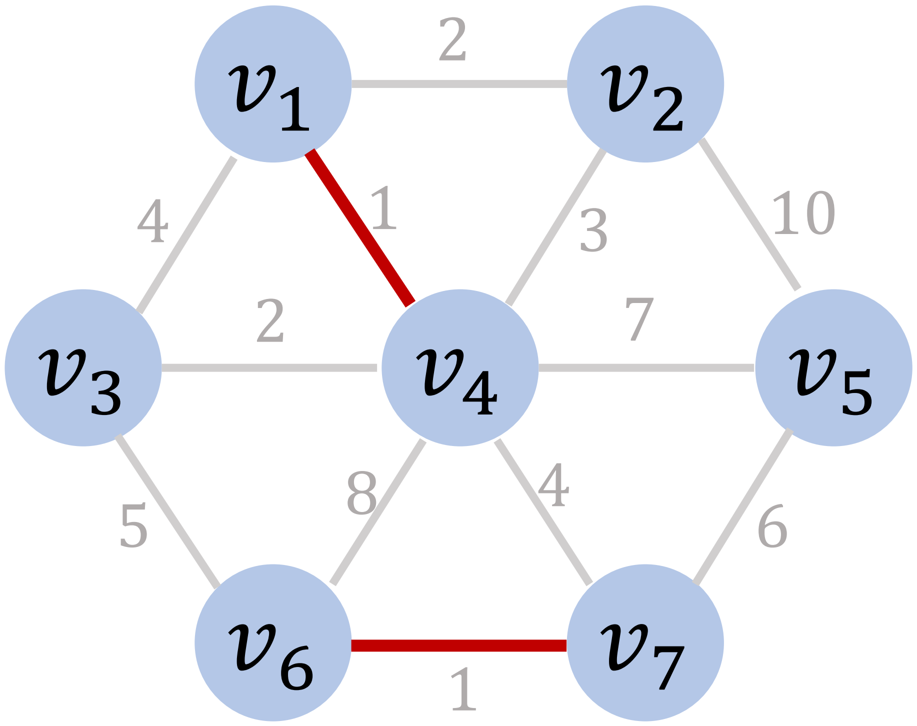


$\mathcal{T} = \{e_{1,4}\}$

- Perform dequeue and get the edge $(6, 7)$.
- v_6 and v_7 are not in the same tree.
- Thus accept edge $(6, 7)$.

Edge	Weight
$(1, 2)$	2
$(3, 4)$	2
$(2, 4)$	3
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 2

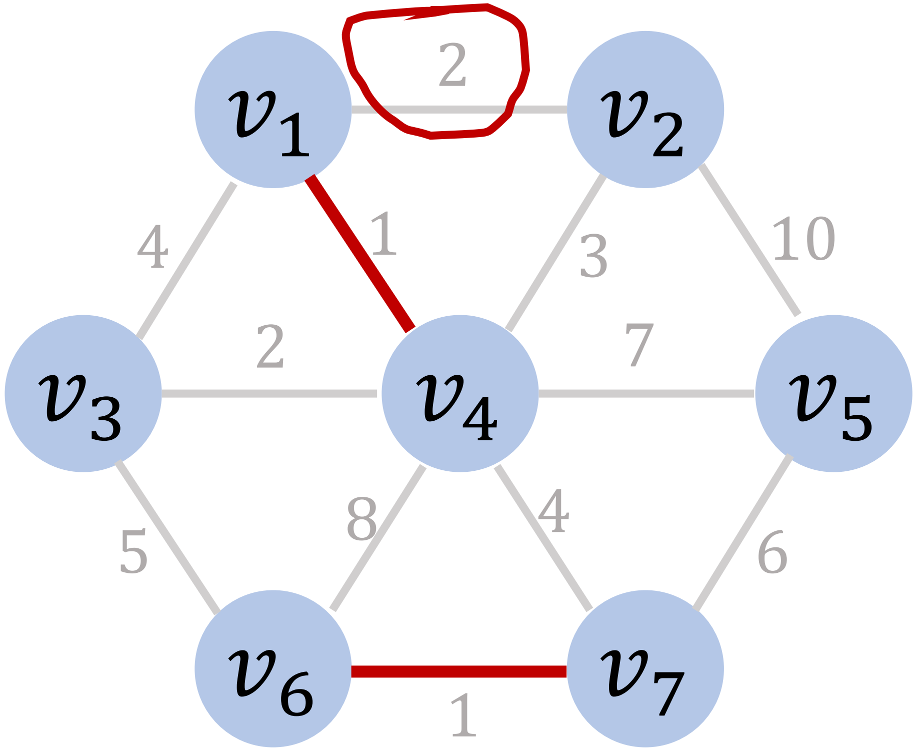


$\mathcal{T} = \{e_{1,4}, e_{6,7}\}$

- Perform dequeue and get the edge $(6, 7)$.
- v_6 and v_7 are not in the same tree.
- Thus accept edge $(6, 7)$.
- Append $(6, 7)$ to $\mathcal{T}$.

Edge	Weight
$(1, 2)$	2
$(3, 4)$	2
$(2, 4)$	3
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 3

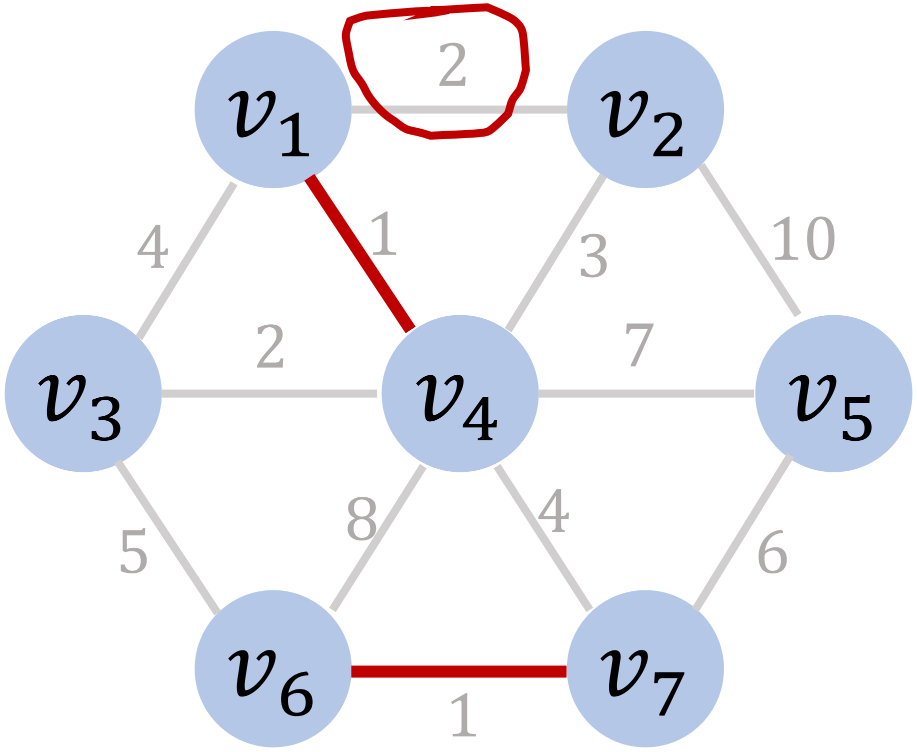


$\mathcal{T} = \{e_{1,4}, e_{6,7}\}$

- Perform dequeue and get the edge $(1, 2)$.

Edge	Weight
$(1, 2)$	2
$(3, 4)$	2
$(2, 4)$	3
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 3

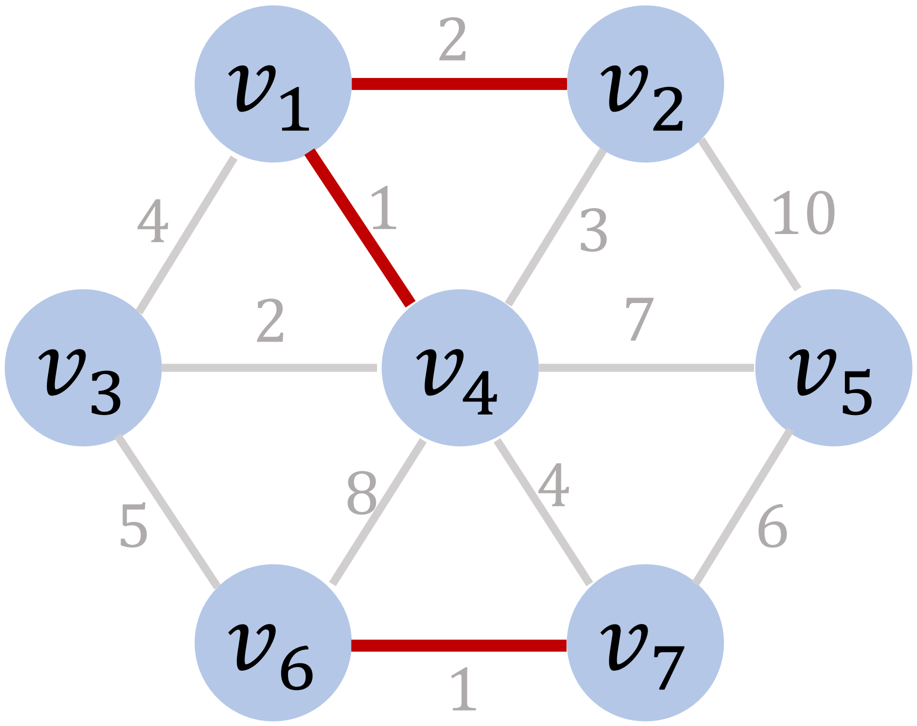


$\mathcal{T} = \{e_{1,4}, e_{6,7}\}$

- Perform dequeue and get the edge $(1, 2)$.
- v_1 and v_2 are not in the same tree.
- Thus accept edge $(1, 2)$.

Edge	Weight
$(3, 4)$	2
$(2, 4)$	3
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 3

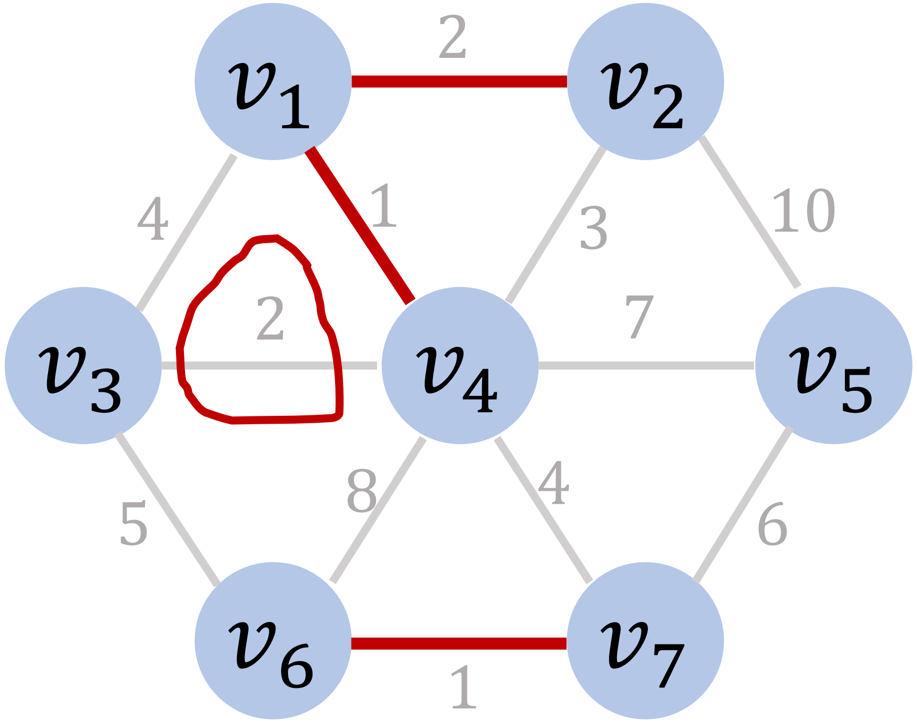


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}\}$

- Perform dequeue and get the edge $(1,2)$.
- v_1 and v_2 are not in the same tree.
- Thus accept edge $(1,2)$.
- Append $(1,2)$ to $\mathcal{T}$.

Edge	Weight
$(3,4)$	2
$(2,4)$	3
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Iteration 4

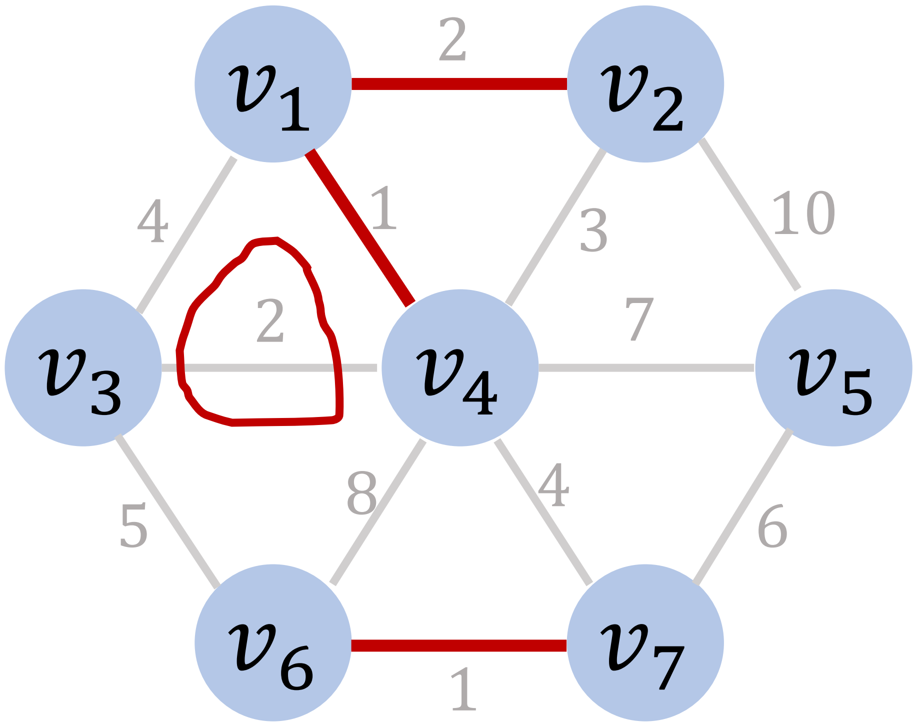


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}\}$

- Perform dequeue and get the edge $(3, 4)$.

Edge	Weight
$(3, 4)$	2
$(2, 4)$	3
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 4

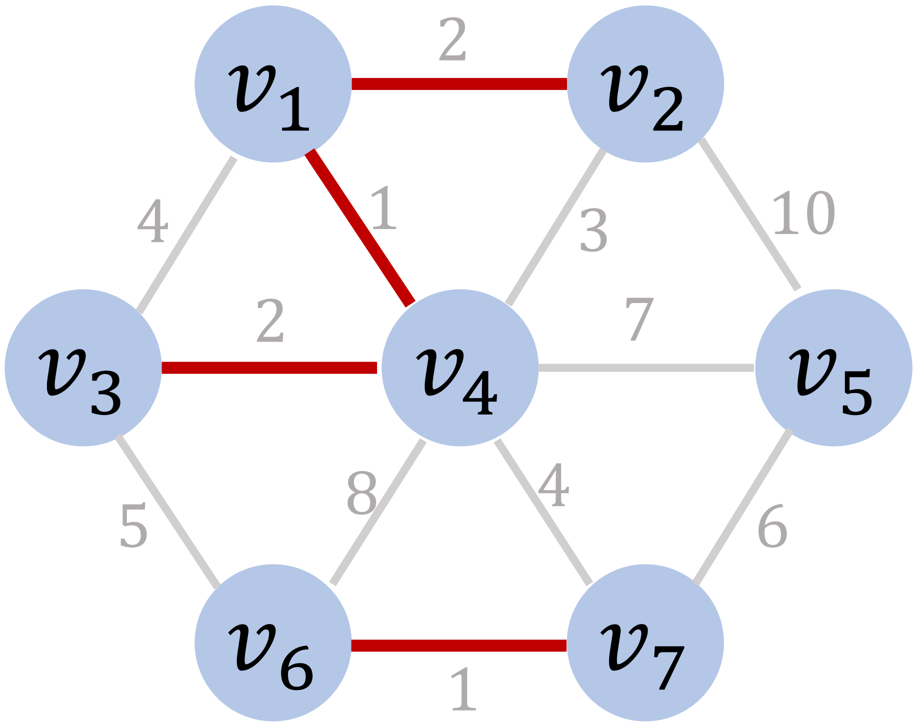


- Perform dequeue and get the edge $(3, 4)$.
- v_3 and v_4 are not in the same tree.
- Thus accept edge $(3, 4)$.

Edge	Weight
$(2, 4)$	3
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}\}$

Iteration 4

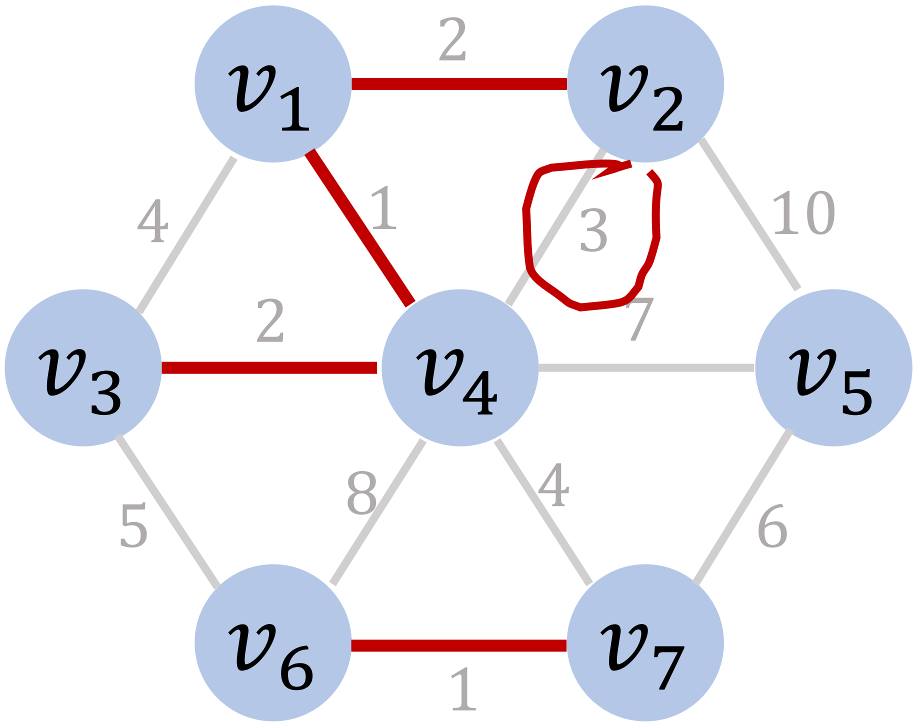


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

- Perform dequeue and get the edge $(3, 4)$.
- v_3 and v_4 are not in the same tree.
- Thus accept edge $(3, 4)$.
- Append $(3, 4)$ to $\mathcal{T}$.

Edge	Weight
$(2, 4)$	3
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 5

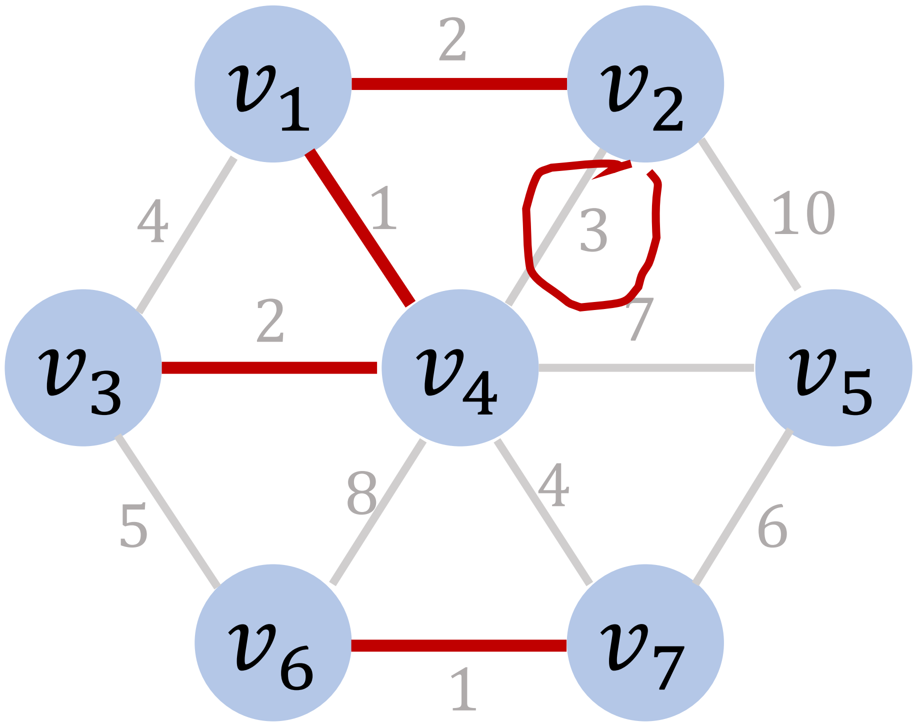


- Perform dequeue and get the edge $(2,4)$.

Edge	Weight
$(2,4)$	3
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

Iteration 5

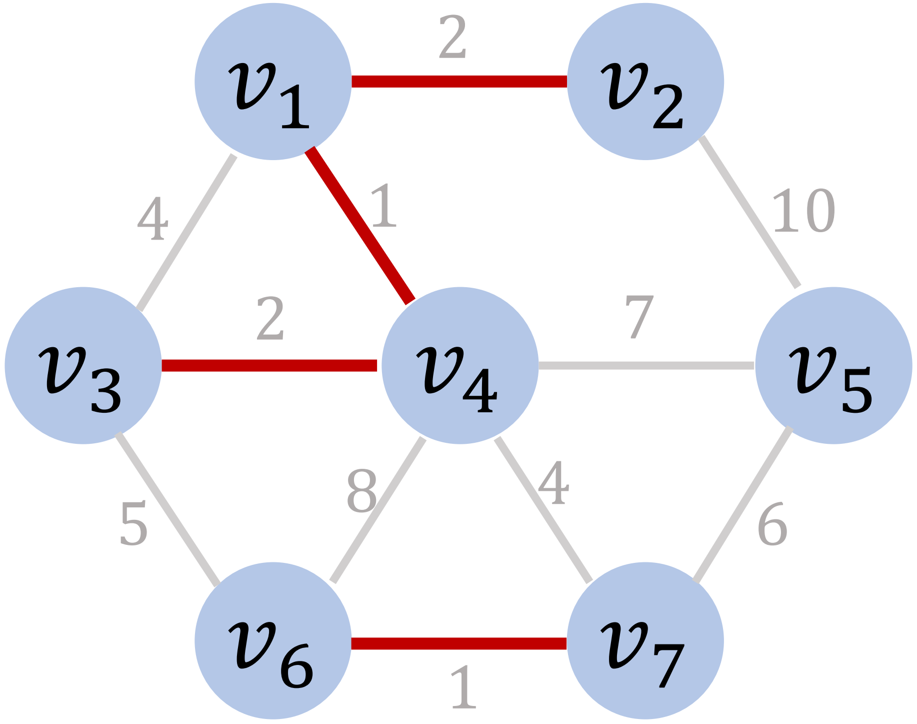


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

- Perform dequeue and get the edge $(2, 4)$.
- v_2 and v_4 are in the same tree.

Edge	Weight
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 5

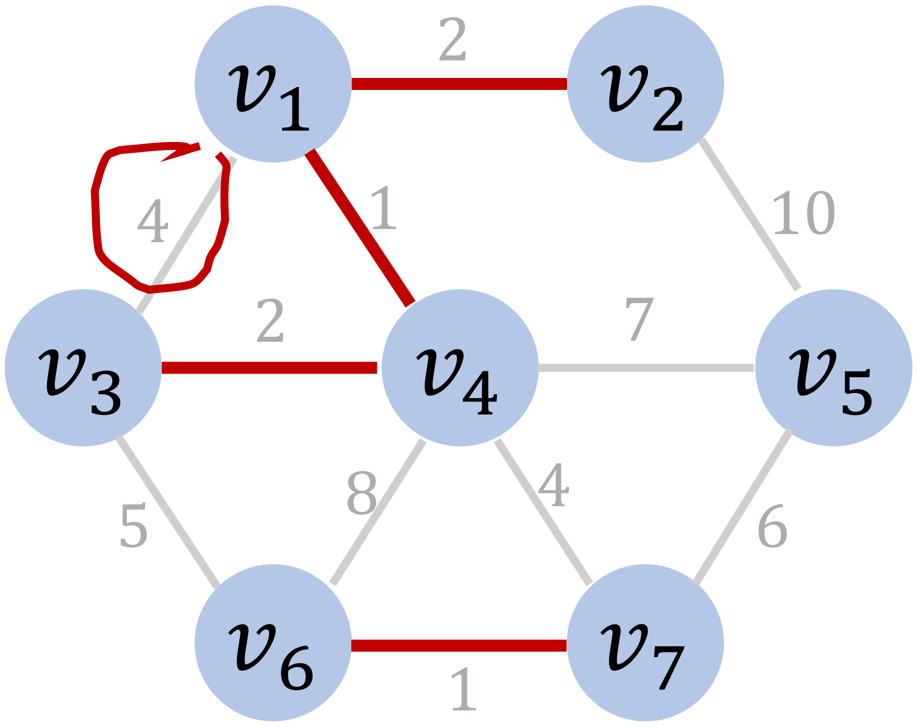


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

- Perform dequeue and get the edge $(2, 4)$.
- v_2 and v_4 are in the same tree.
- Thus reject edge $(2, 4)$.

Edge	Weight
$(1, 3)$	4
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 6

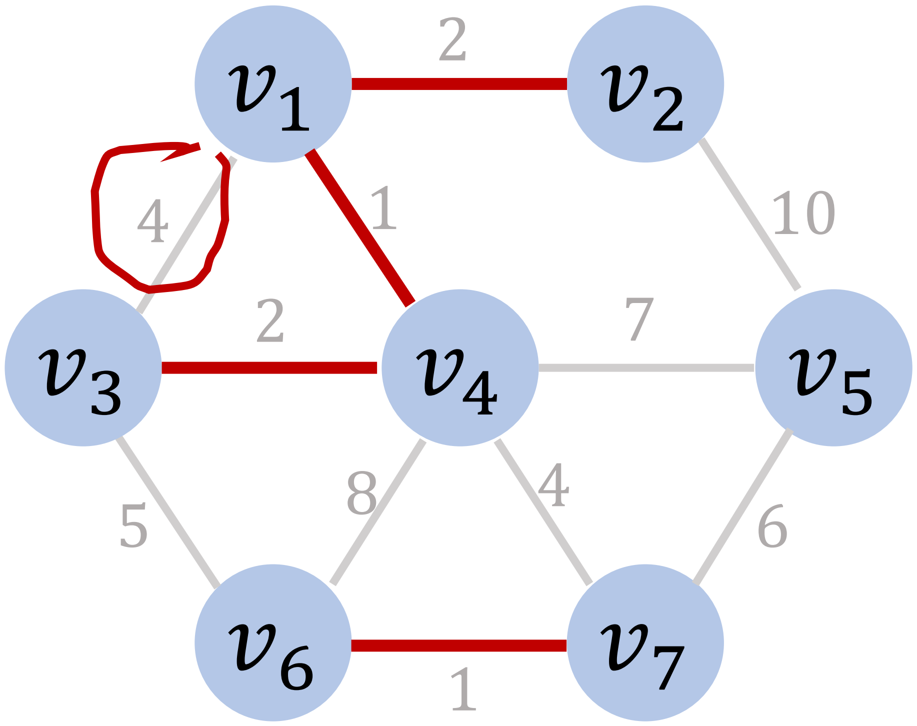


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

- Perform dequeue and get the edge $(1,3)$.

Edge	Weight
$(1,3)$	4
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Iteration 6

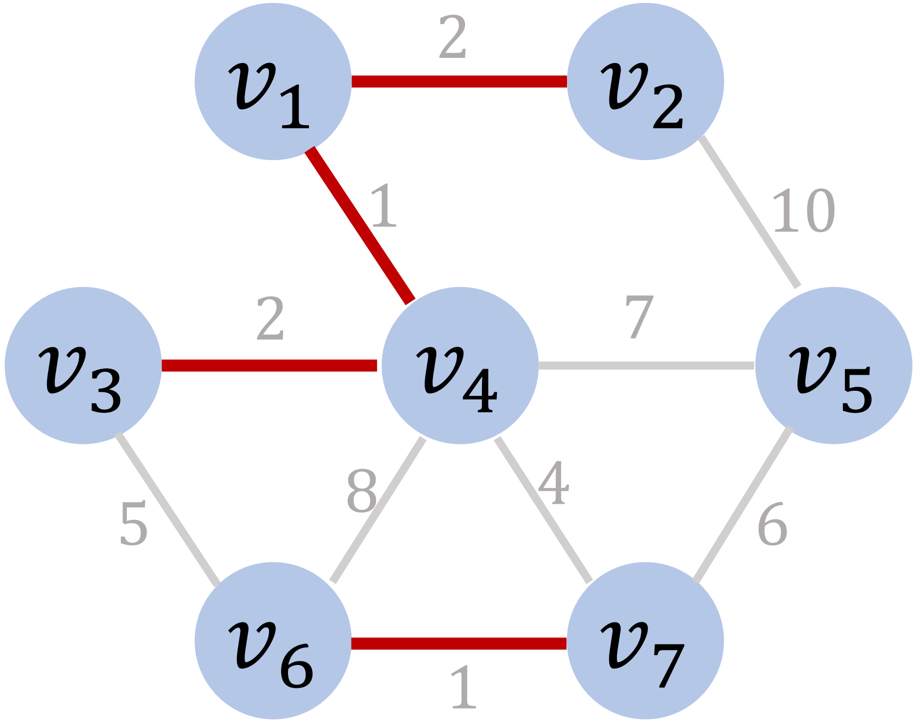


- Perform dequeue and get the edge $(1, 3)$.
- v_1 and v_3 are in the same tree.

Edge	Weight
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

Iteration 6

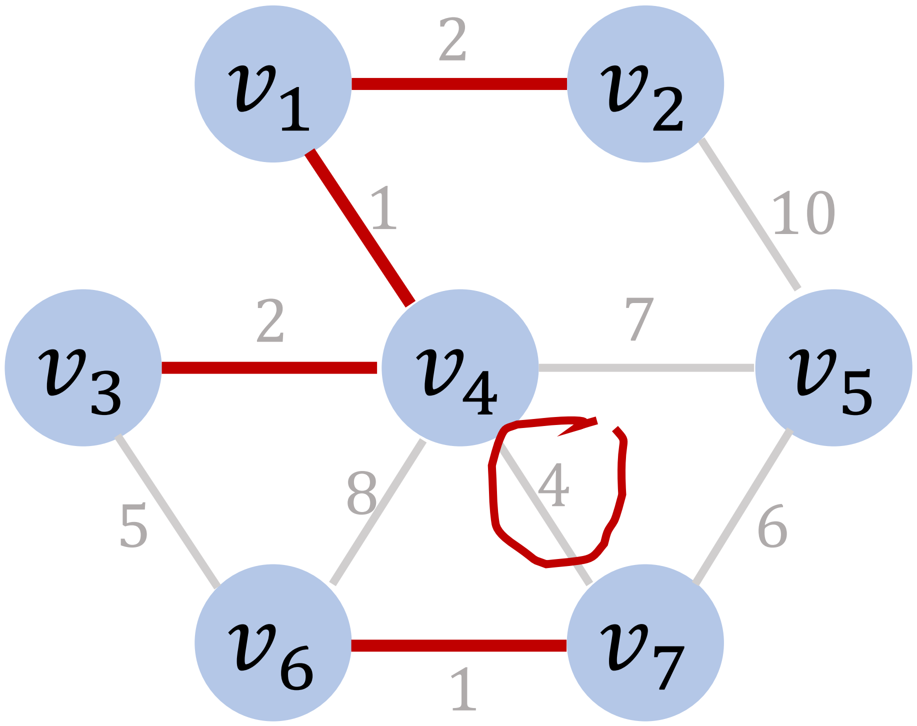


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

- Perform dequeue and get the edge $(1, 3)$.
- v_1 and v_3 are in the same tree.
- Thus reject edge $(1, 3)$.

Edge	Weight
$(4, 7)$	4
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 7

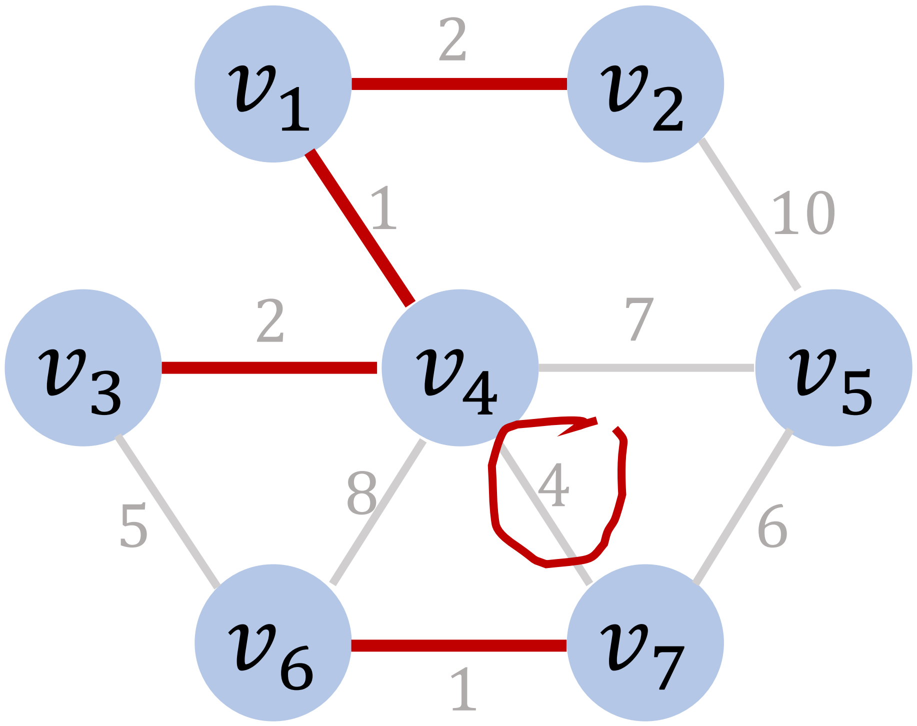


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

- Perform dequeue ueue and get the edge $(4,7)$.

Edge	Weight
$(4,7)$	4
$(3,6)$	5
$(5,7)$	6
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Iteration 7

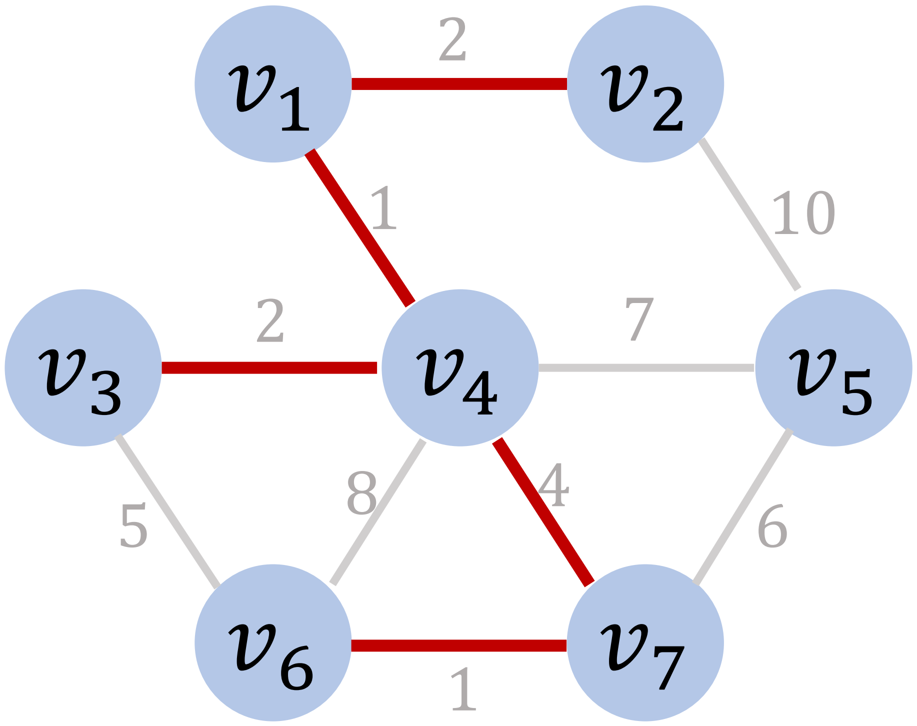


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}\}$

- Perform dequeue and get the edge $(4, 7)$.
- v_4 and v_7 are not in the same tree.
- Thus accept edge $(4, 7)$.

Edge	Weight
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 7

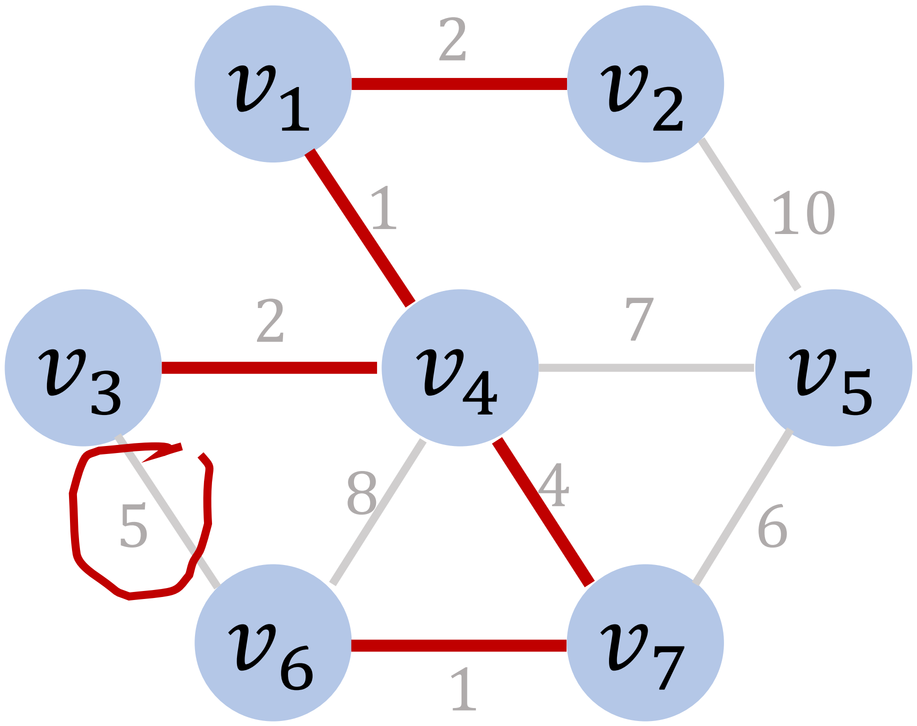


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}, e_{4,7}\}$

- Perform dequeue and get the edge $(4, 7)$.
- v_4 and v_7 are not in the same tree.
- Thus accept edge $(4, 7)$.
- Append $(4, 7)$ to $\mathcal{T}$.

Edge	Weight
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 8

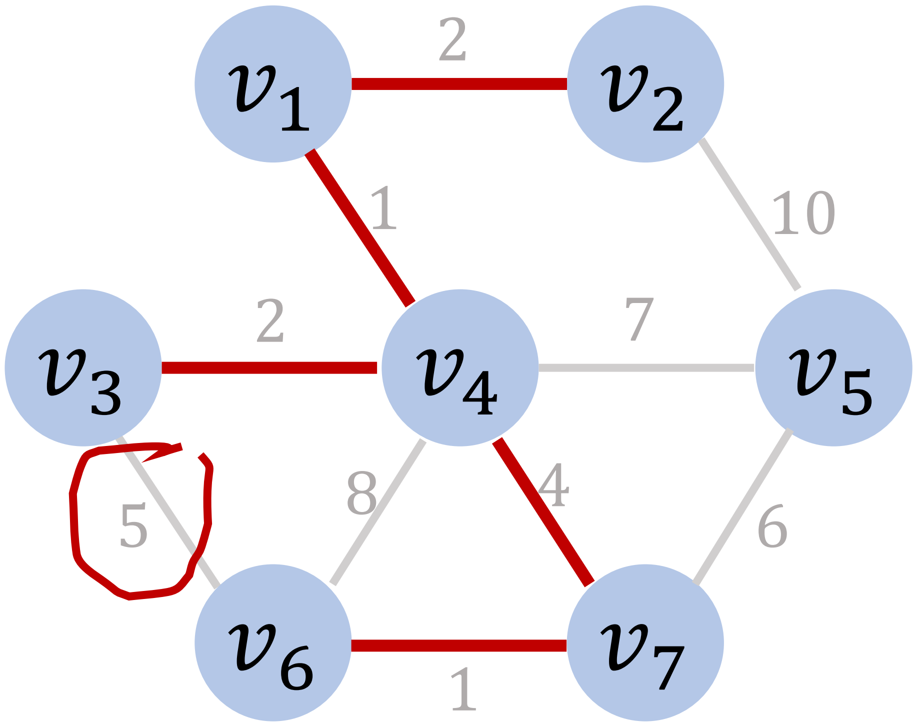


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}, e_{4,7}\}$

- Perform dequeue and get the edge $(3, 6)$.

Edge	Weight
$(3, 6)$	5
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 8

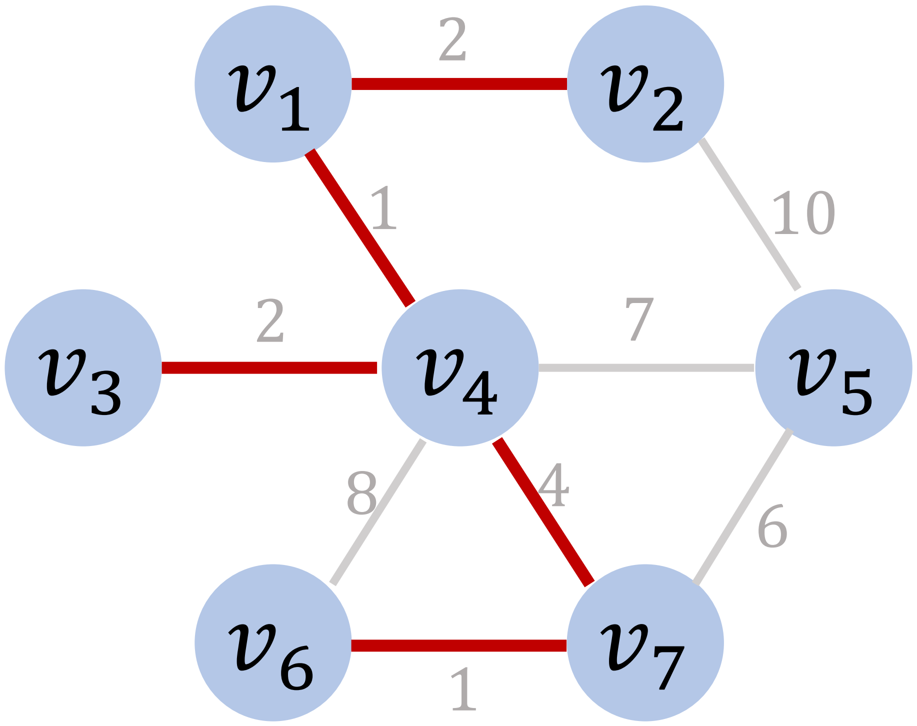


- Perform dequeue and get the edge $(3, 6)$.
- v_3 and v_6 are not in the same tree.

Edge	Weight
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}, e_{4,7}\}$

Iteration 8

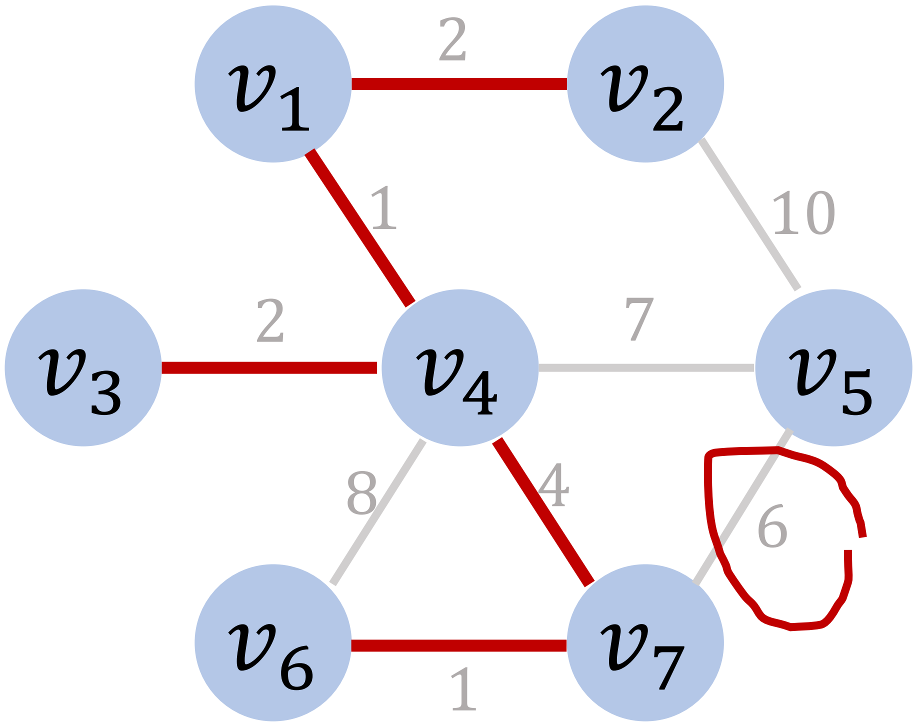


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}, e_{4,7}\}$

- Perform dequeue and get the edge $(3, 6)$.
- v_3 and v_6 are not in the same tree.
- Thus reject edge $(3, 6)$.

Edge	Weight
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 9

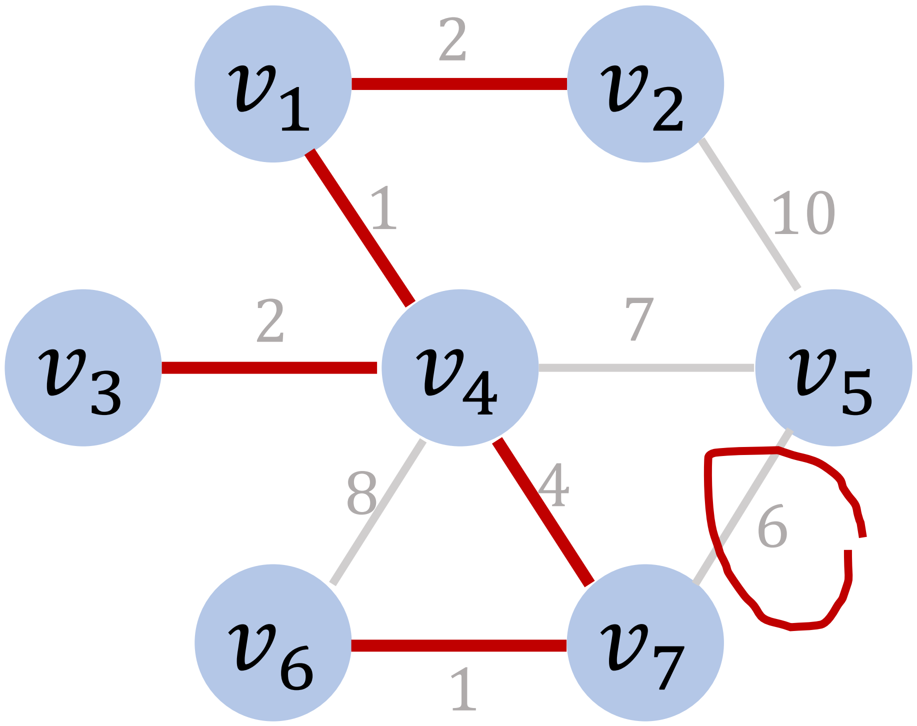


- Perform dequeue and get the edge $(5, 7)$.

Edge	Weight
$(5, 7)$	6
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}, e_{4,7}\}$

Iteration 9

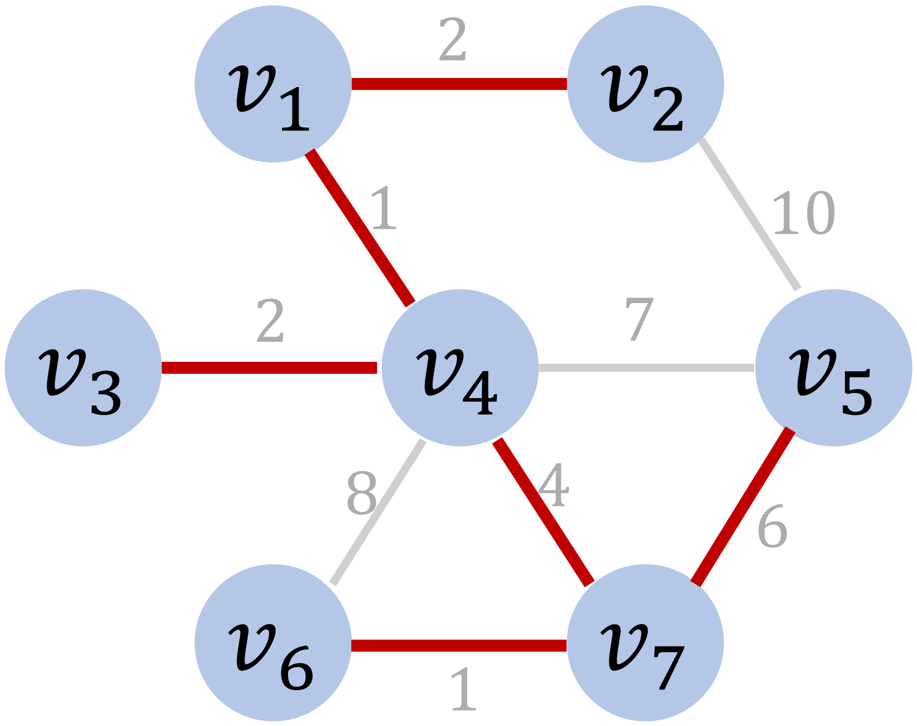


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}, e_{4,7}\}$

- Perform dequeue and get the edge $(5, 7)$.
- v_5 and v_7 are not in the same tree.
- Thus accept edge $(5, 7)$.

Edge	Weight
$(4, 5)$	7
$(4, 6)$	8
$(2, 5)$	10

Iteration 9

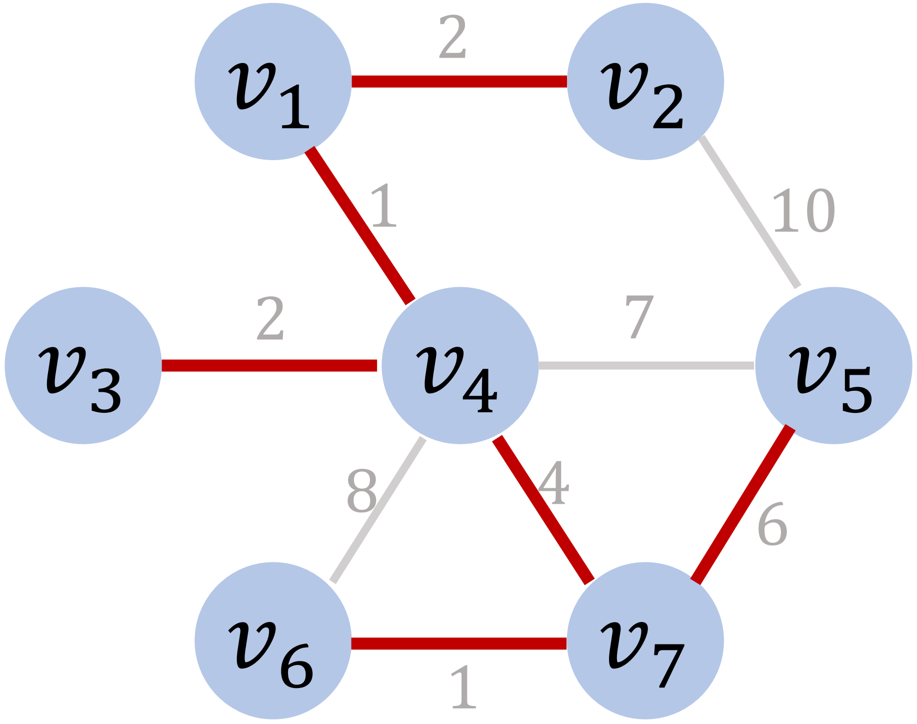


$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}, e_{4,7}, e_{5,7}\}$

- Perform dequeue and get the edge $(5,7)$.
- v_5 and v_7 are not in the same tree.
- Thus accept edge $(5,7)$.
- Append $(5,7)$ to $\mathcal{T}$.

Edge	Weight
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

End of Procedure



$$\mathcal{T} = \{e_{1,4}, e_{6,7}, e_{1,2}, e_{3,4}, e_{4,7}, e_{5,7}\}$$

- All the vertices are connected.
- Return the edges $\mathcal{T}$.

Edge	Weight
$(4,5)$	7
$(4,6)$	8
$(2,5)$	10

Kruskal's Algorithm

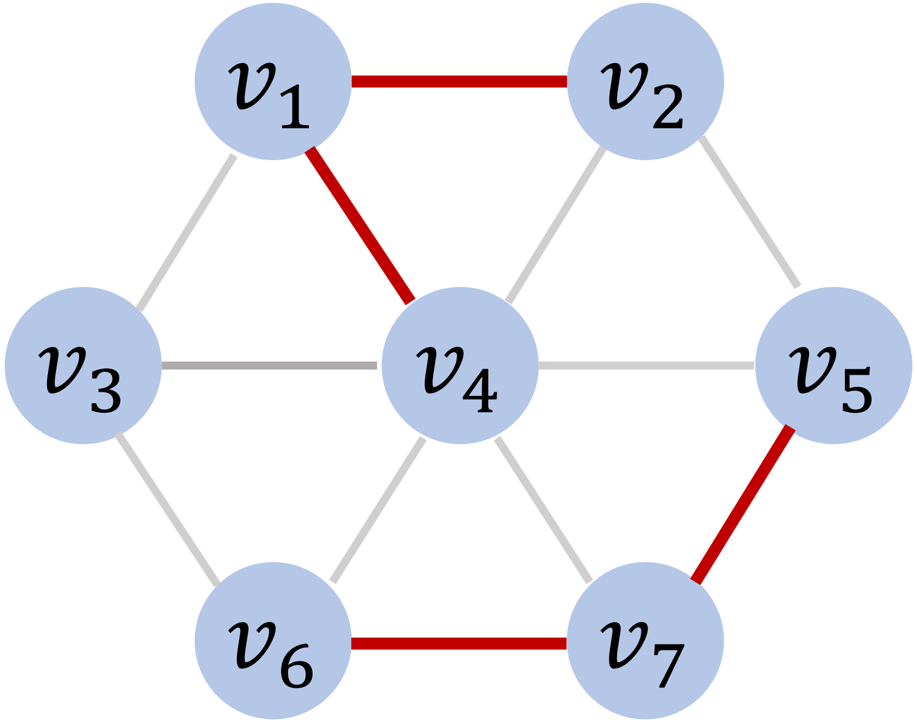
1. Put all the edges of the input graph in a queue.
2. Sort the queue so that the weights are in the ascending order.
3. Let set $\mathcal{T}$ (which stores selected edges) be the empty set.

Kruskal's Algorithm

1. Put all the edges of the input graph in a queue.
2. Sort the queue so that the weights are in the ascending order.
3. Let set $\mathcal{T}$ (which stores selected edges) be the empty set.
4. While $|\mathcal{T}| \leq n - 1$:
   a. Get an edge: $e_{uv} \leftarrow \text{dequeue}(\,)$.
   b. If u and v are in different trees, then add e_{uv} to $\mathcal{T}$ and merge the two trees.
5. Return $\mathcal{T}$.

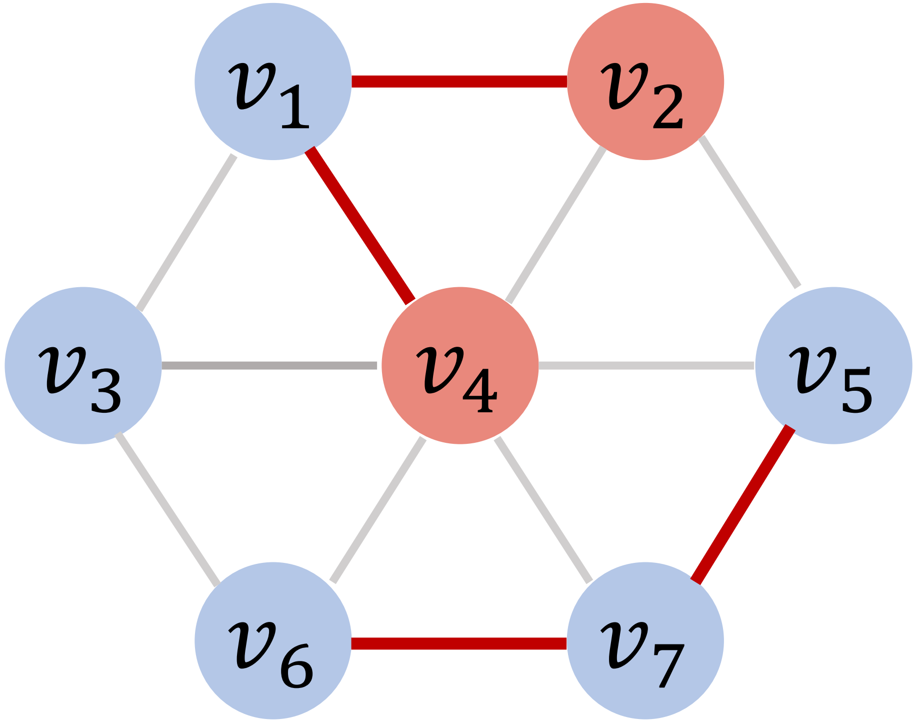
How to maintain the forest?

Using Disjoint Sets Data Structure



Question 1: How to decide whether two vertices are in the same tree?

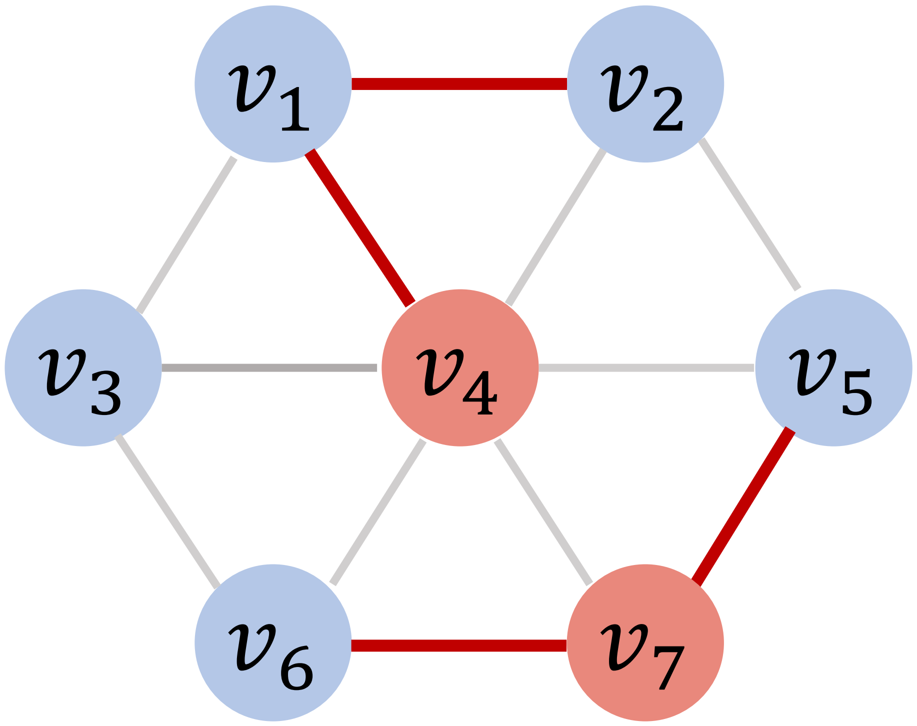
Using Disjoint Sets Data Structure



Question 1: How to decide whether two vertices are in the same tree?

- Are v_2 and v_4 in the same tree?

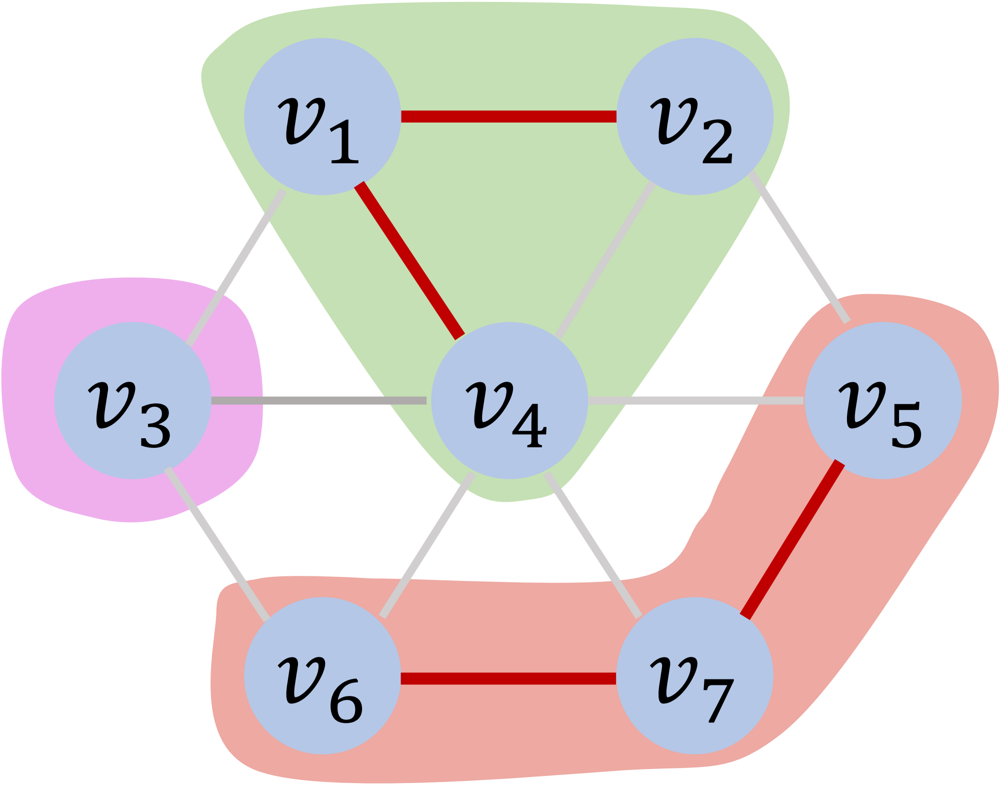
Using Disjoint Sets Data Structure



Question 1: How to decide whether two vertices are in the same tree?

- Are v_4 and v_7 in the same tree?

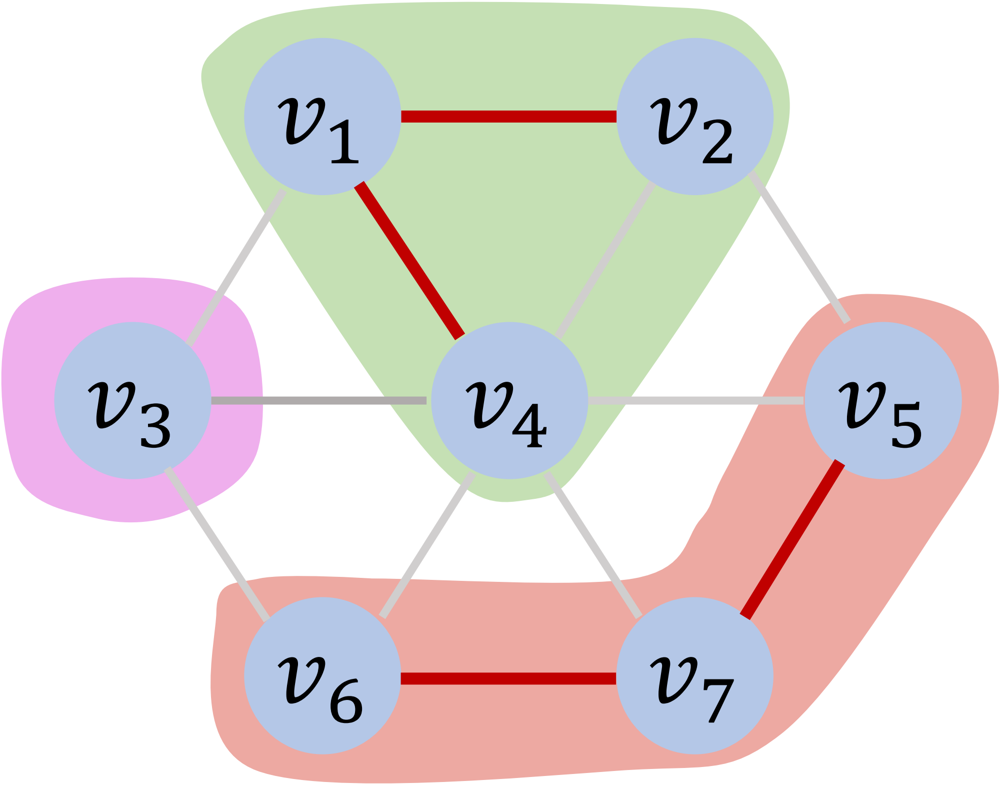
Using Disjoint Sets Data Structure



Question 1: How to decide whether two vertices are in the same tree?

- Using disjoint sets data structure.
- Put vertices of a tree in the same set.

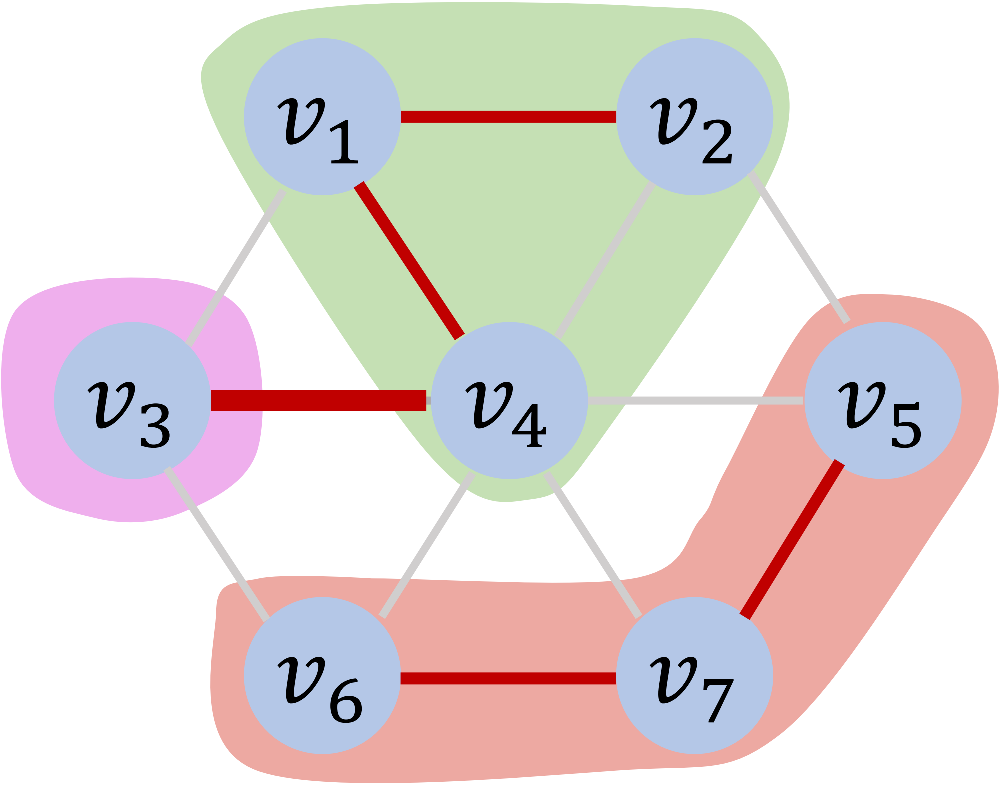
Using Disjoint Sets Data Structure



Question 1: How to decide whether two vertices are in the same tree?

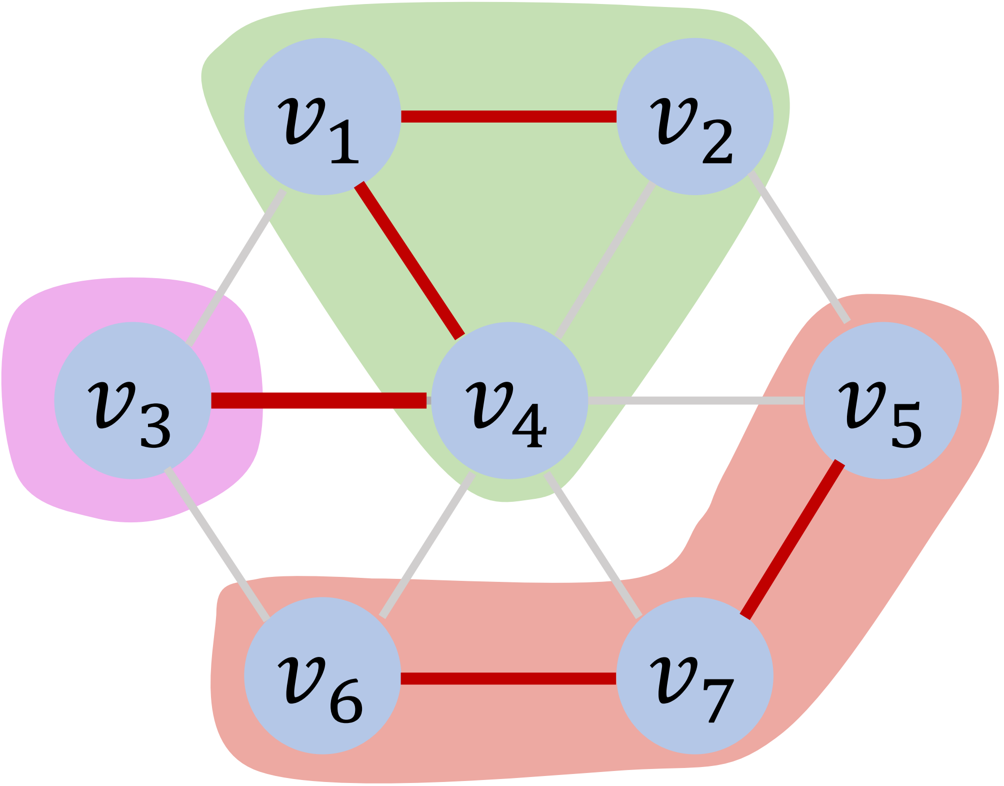
- Using disjoint sets data structure.
- Put vertices of a tree in the same set.
- Deciding whether two vertices belong to the same set costs near $O(1)$ time.

Using Disjoint Sets Data Structure



Question 2: How to merge two trees?

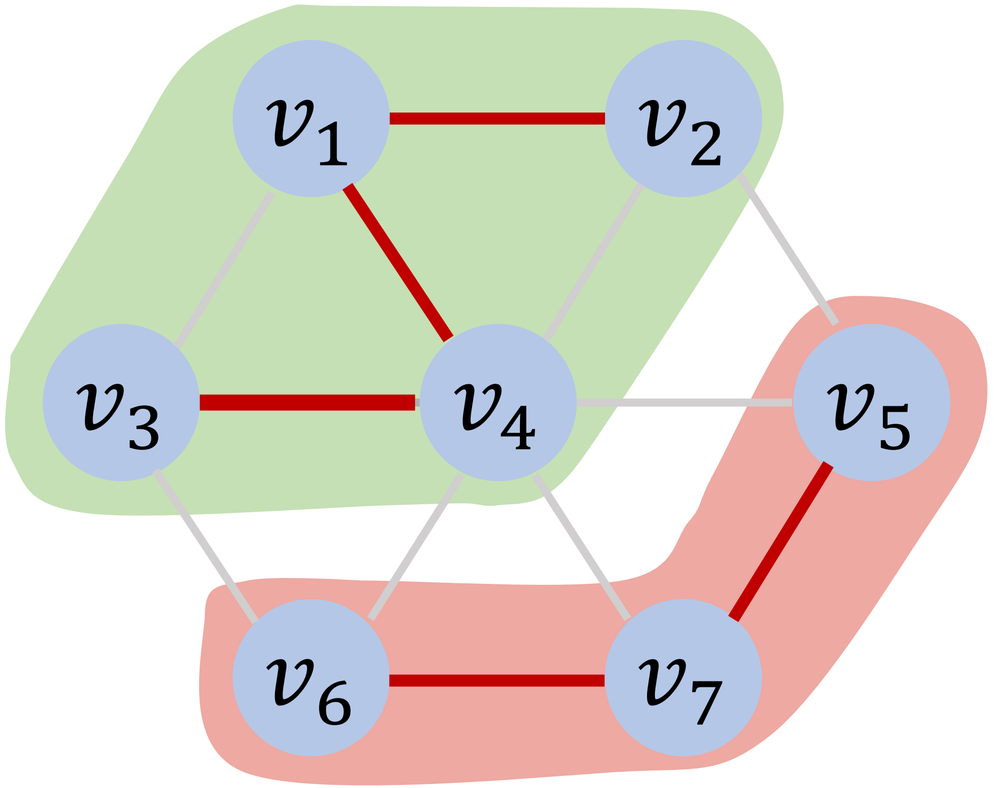
Using Disjoint Sets Data Structure



Question 2: How to merge two trees?

- Union the two sets.

Using Disjoint Sets Data Structure



Question 2: How to merge two trees?

- Union the two sets.
- Union costs near $O(1)$ time.

Time Complexity

Overall time complexity is $O(m \cdot \log m)$. $(m =$ #edges.)

- Sorting the edges: $O(m \cdot \log m)$ time complexity.
- At most m iterations.
- Per-iteration time complexity is near $O(1)$.
  - Deciding whether two vertices belong to the same tree: near $O(1)$ time.
  - Merge two trees: near $O(1)$ time.

Thank You!