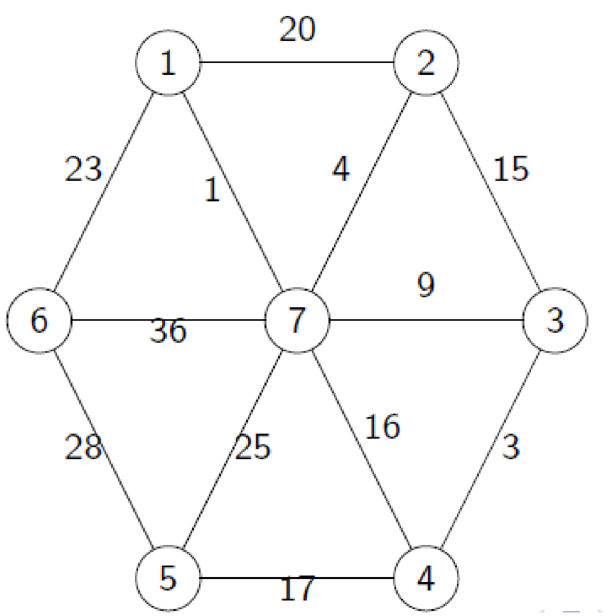
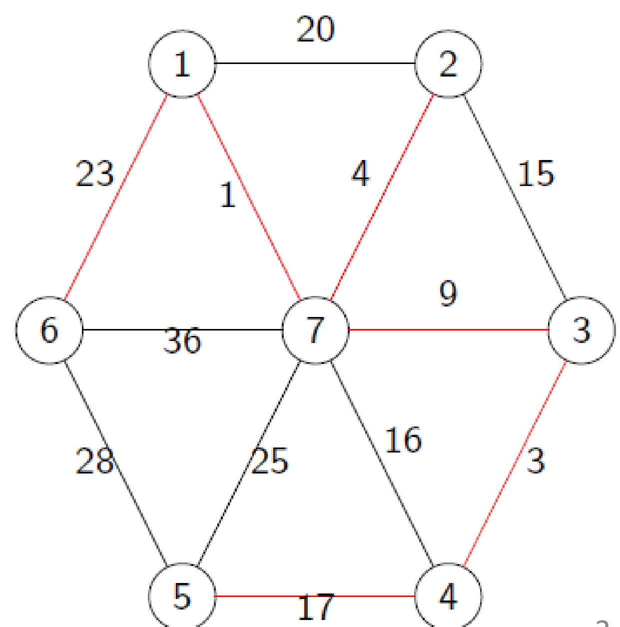
Minimum Spanning Trees

Adapted from notes by Mahesh Viswanathan (UIUC)

MST: the problem

- Input: Connected graph $G = (V, E)$ with edge costs
- Goal: find $T \subseteq E$ such that (V, T) is connected and the total cost of all edges in T is the smallest
  - T is called the Minimum spanning tree of G





Practical application of MST

- Network design
  - Telephone, computer, road etc.
  - A set of locations and find the cheapest way of connecting them
  - Must be a spanning tree
    - Need to connect all nodes
    - Must be a tree otherwise we can remove some edges and still be connected and cheaper
- Other applications
  - Approximations to NP-hard problems like traveling salesman problem
  - Clustering in machine learning
  - …

Greedy Template

T is empty (* T will store edges of a MST*)

While T is not spanning tree yet (* $|T| \leq |V| - 1$ *)
    select $e \in E$ to add to T according to a greedy criterion

Return T

How should we choose the edge to add to T?

We will present two different greedy criteria and then show a unifying proof for the correctness

Kruskal's algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

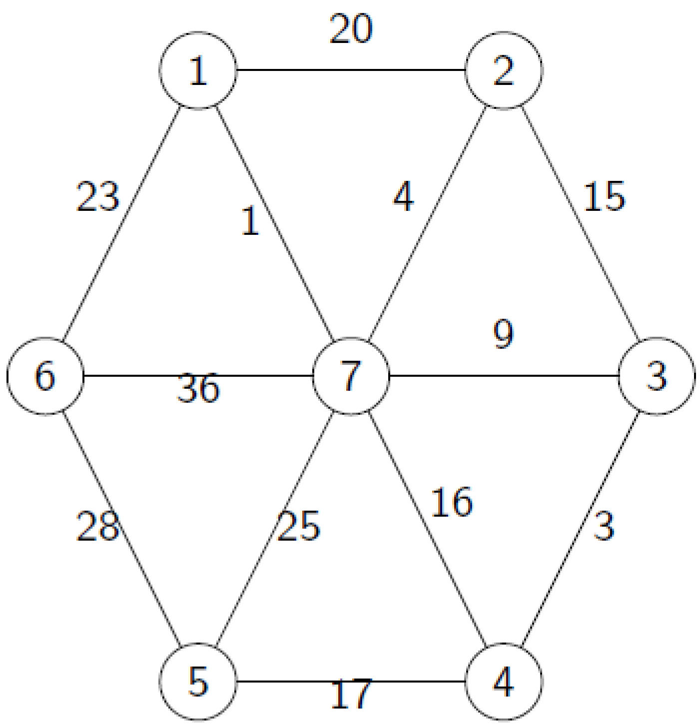


Figure: Graph G

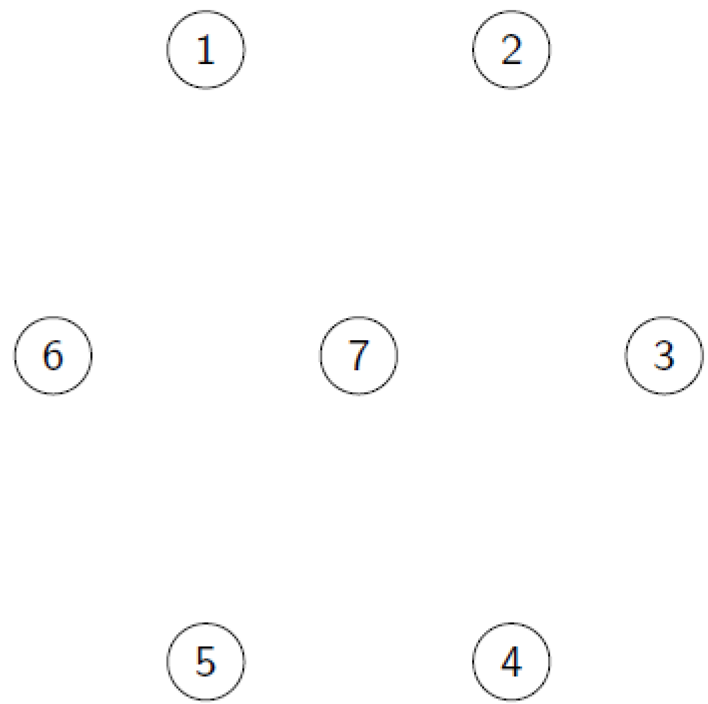


Figure: MST of G

Kruskal's algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

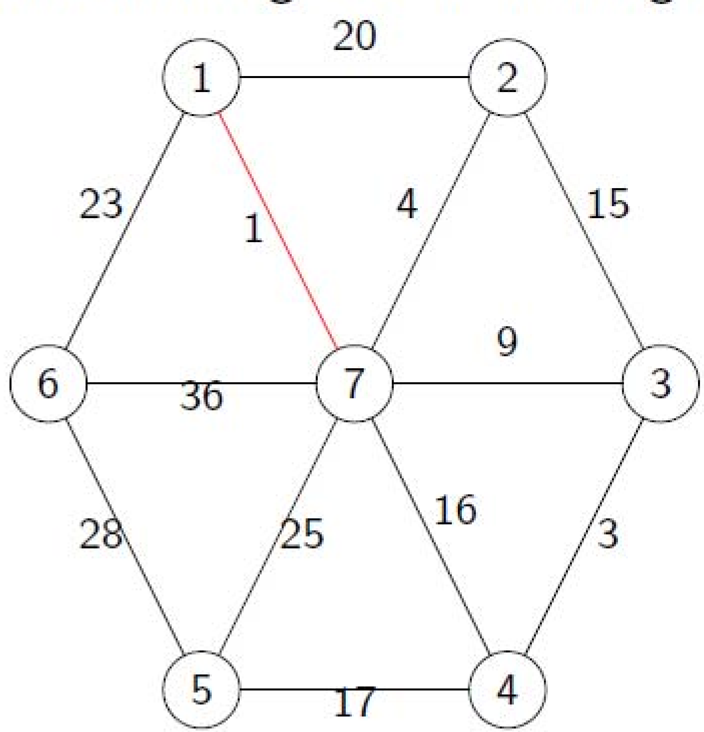


Figure: Graph G

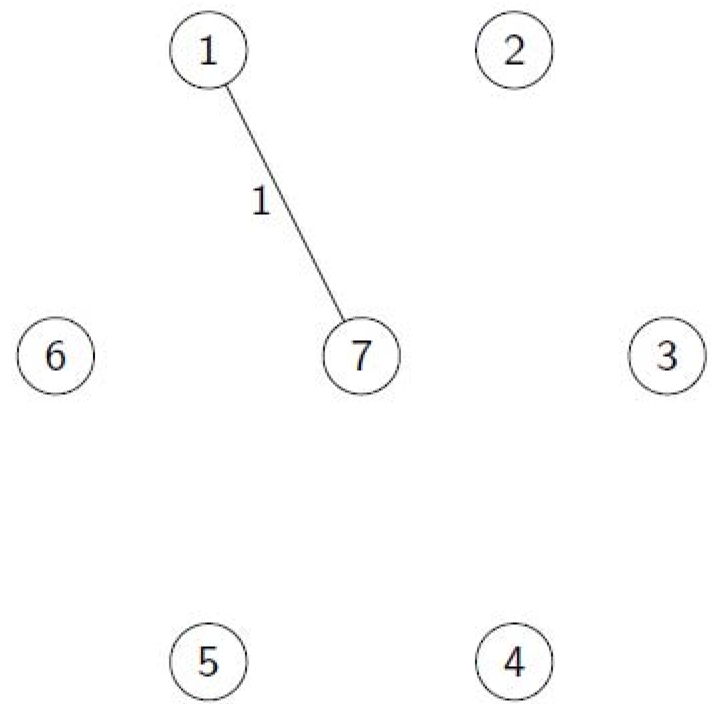


Figure: MST of G

Kruskal's algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

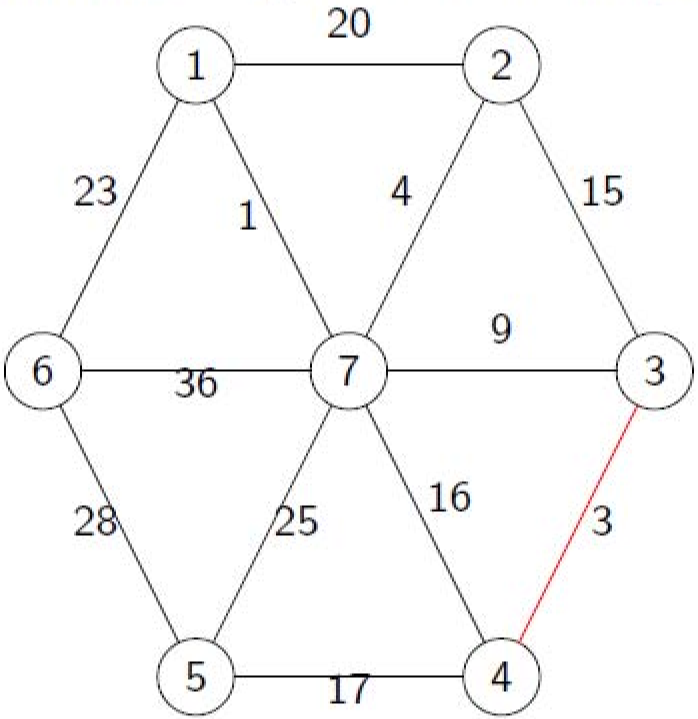


Figure: Graph G

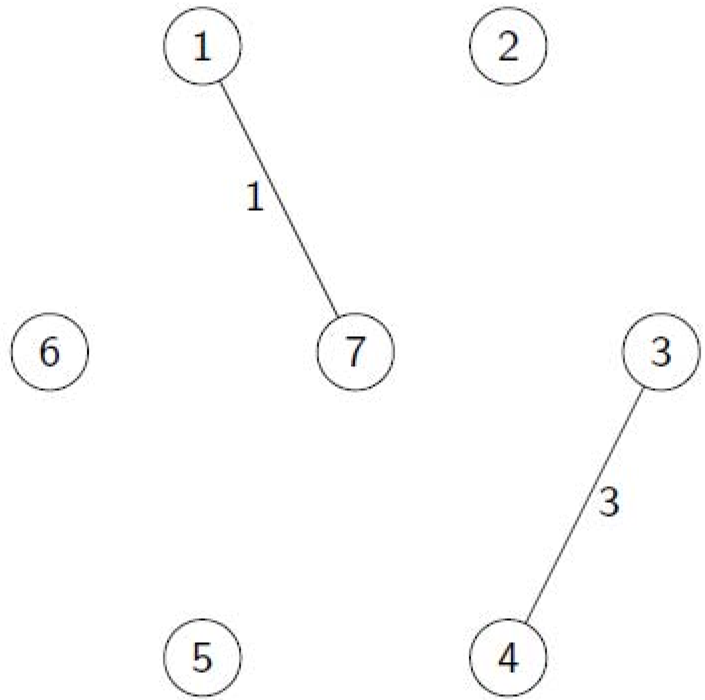


Figure: MST of G

Kruskal's algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

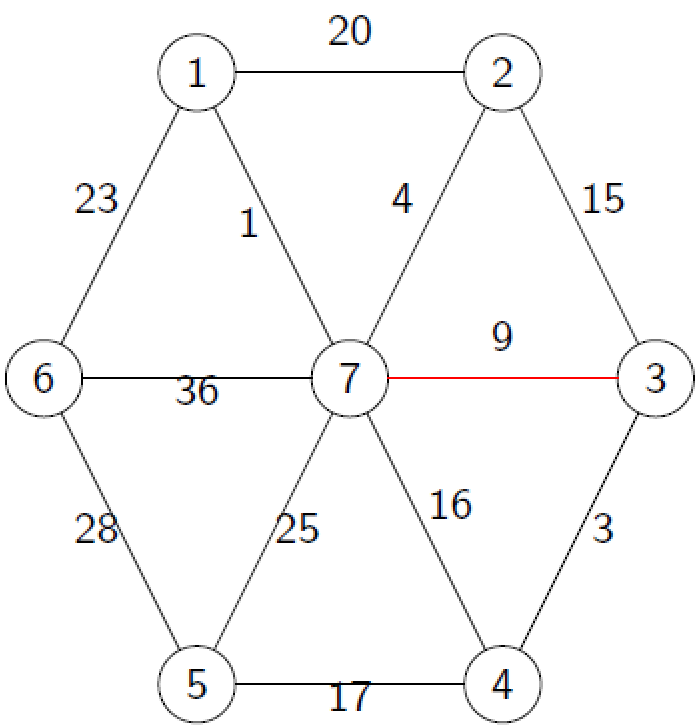


Figure: Graph G

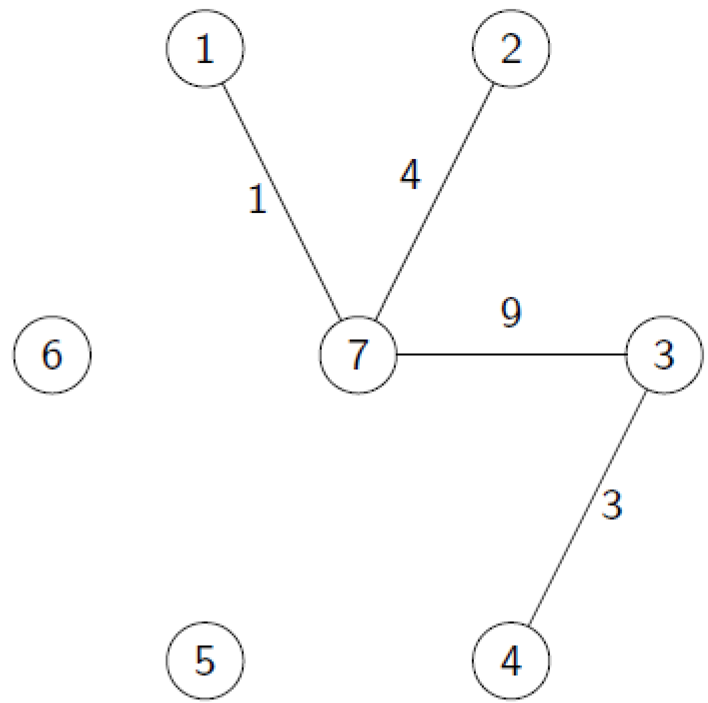


Figure: MST of G

Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

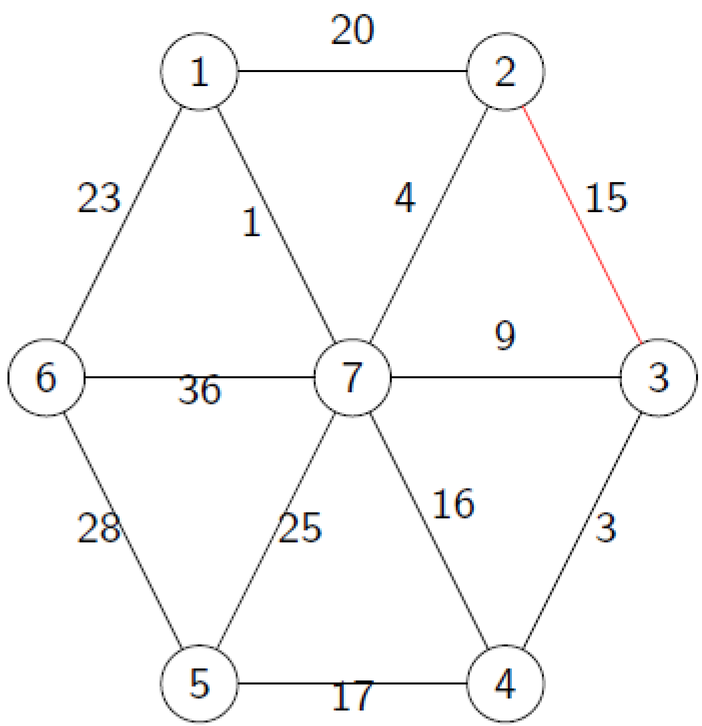


Figure: Graph G

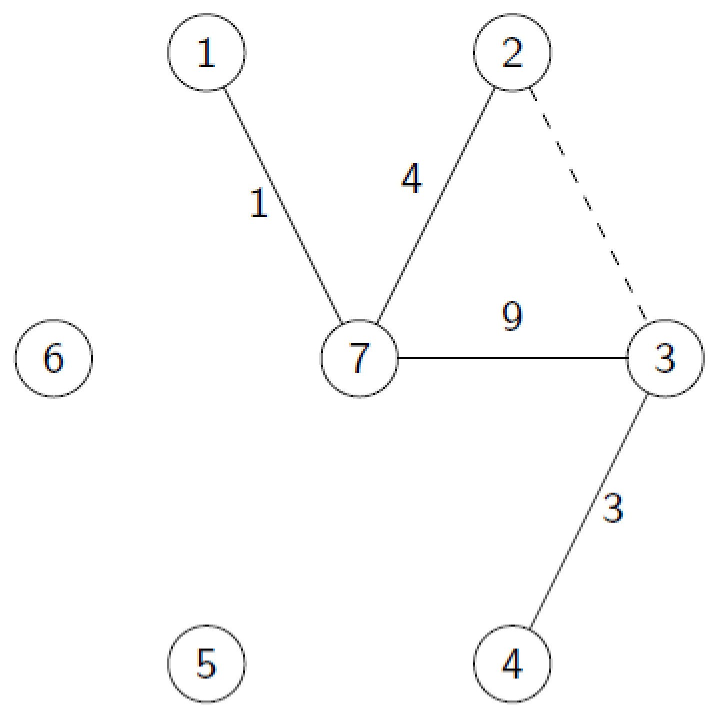


Figure: MST of G

Kruskal's algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

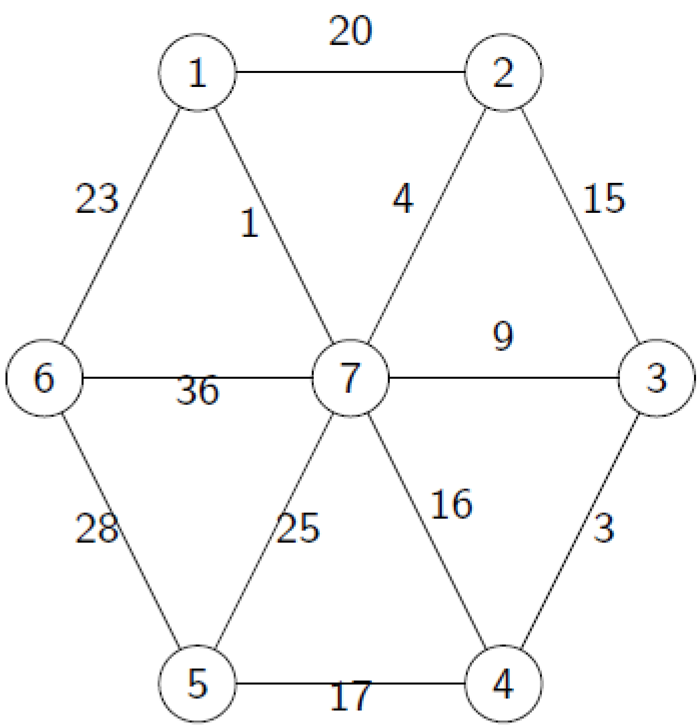


Figure: Graph G

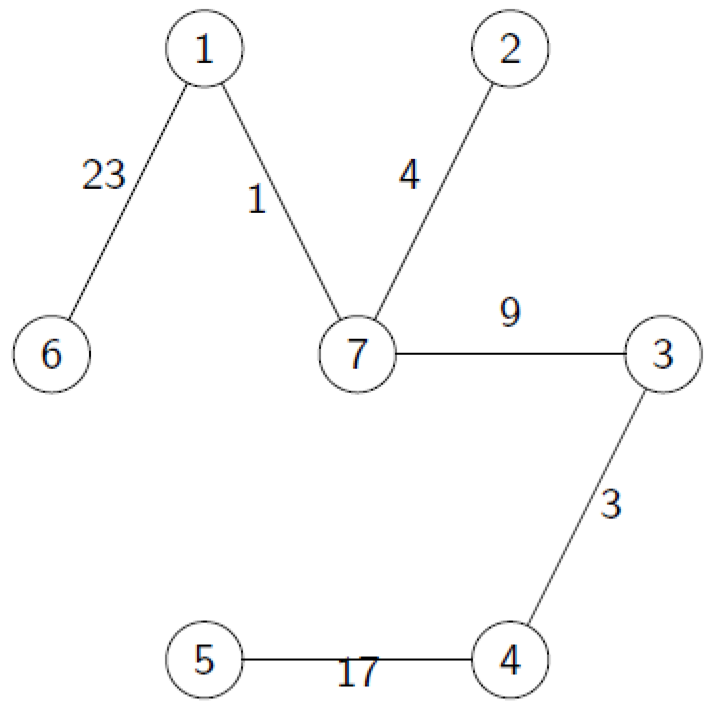


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

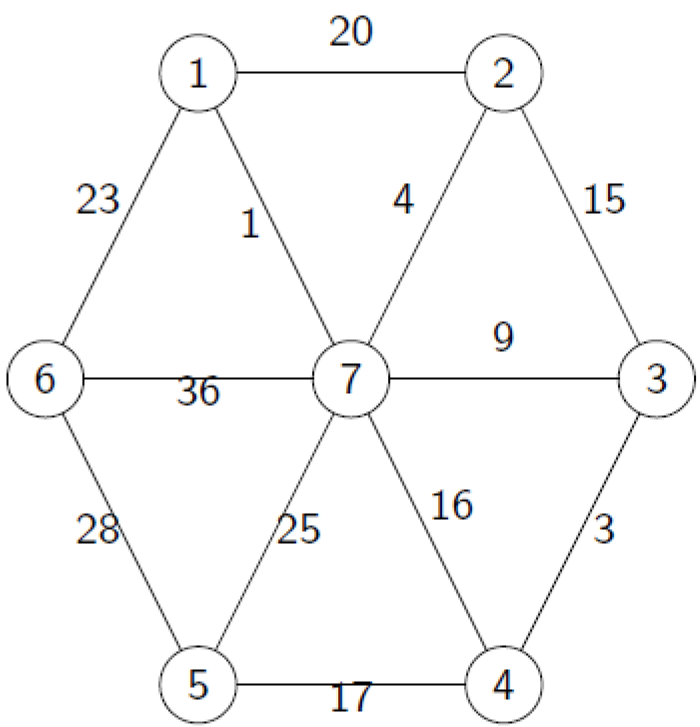


Figure: Graph G

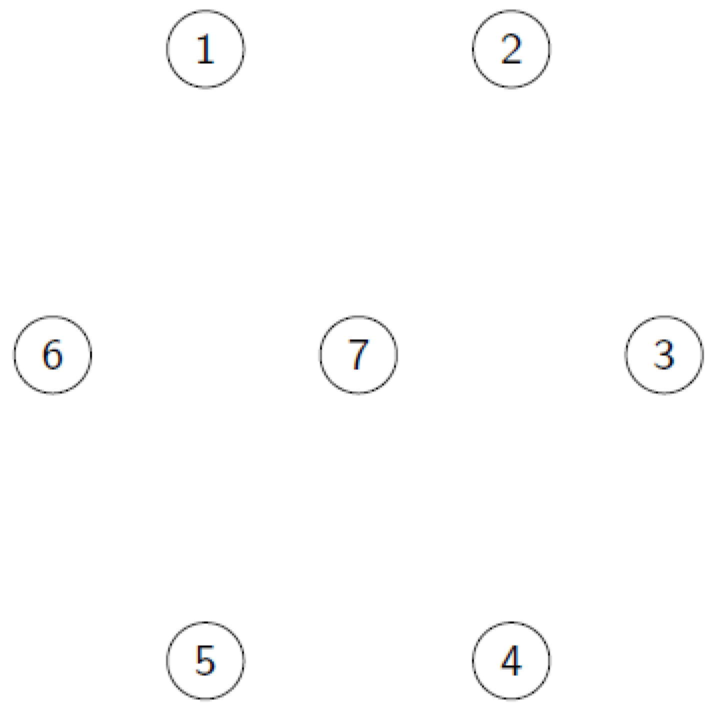


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

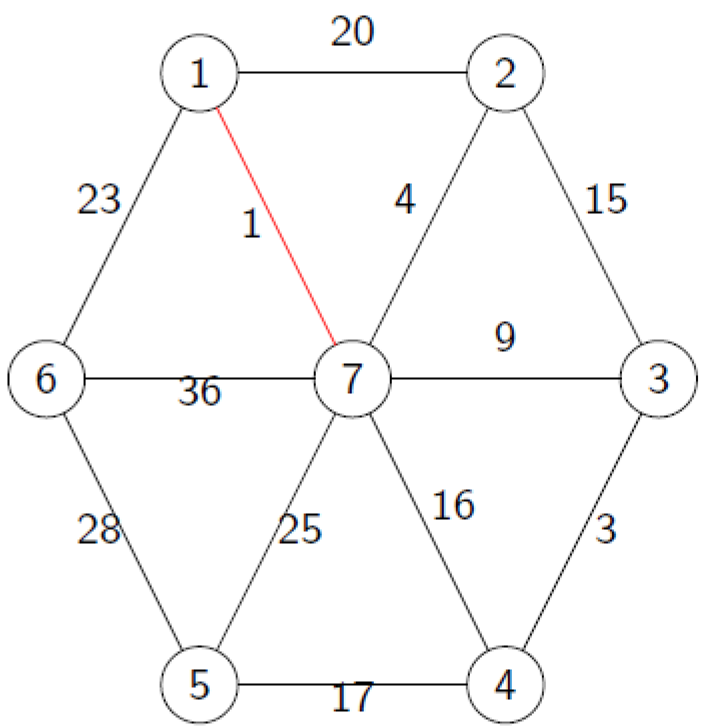


Figure: Graph G

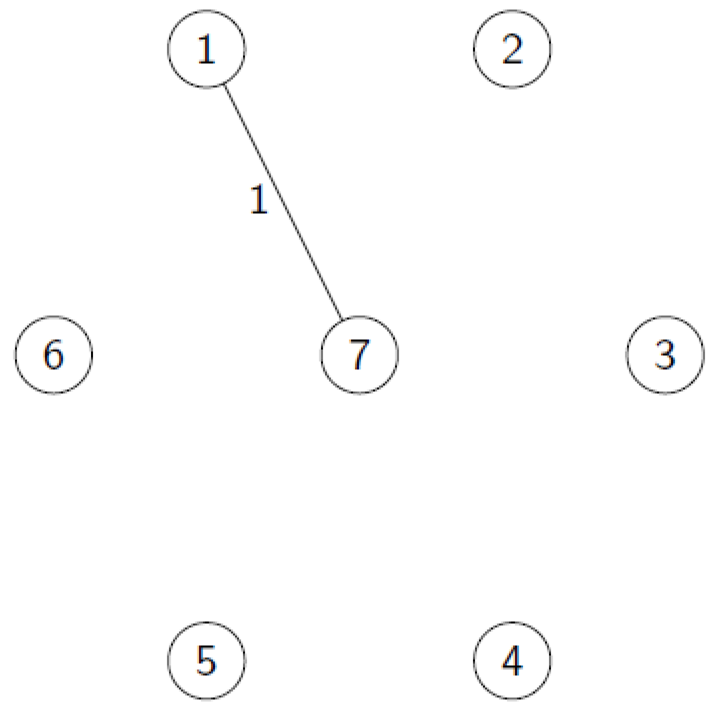


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

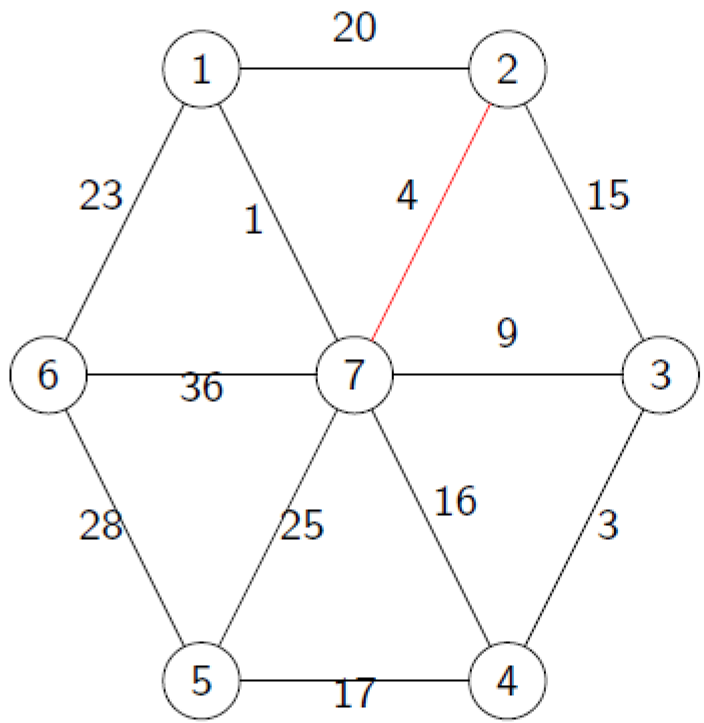


Figure: Graph G

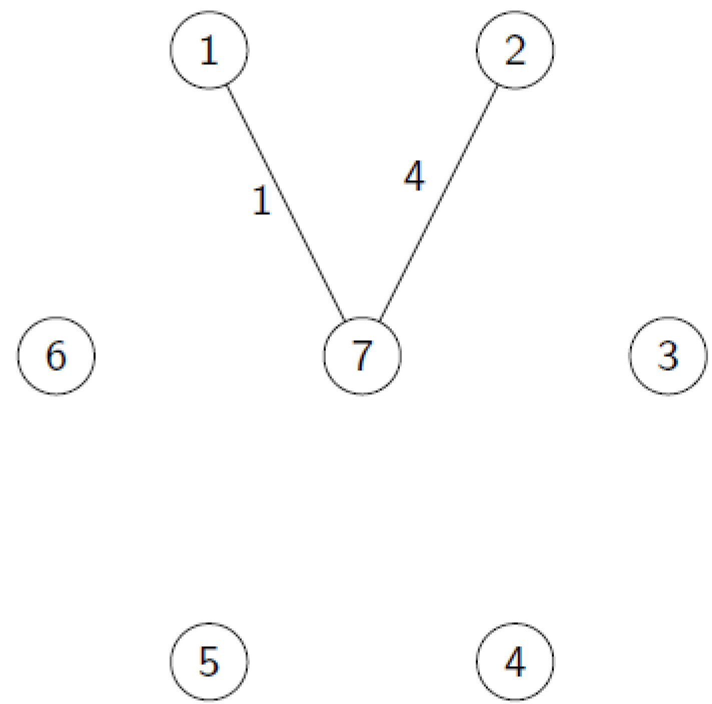


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

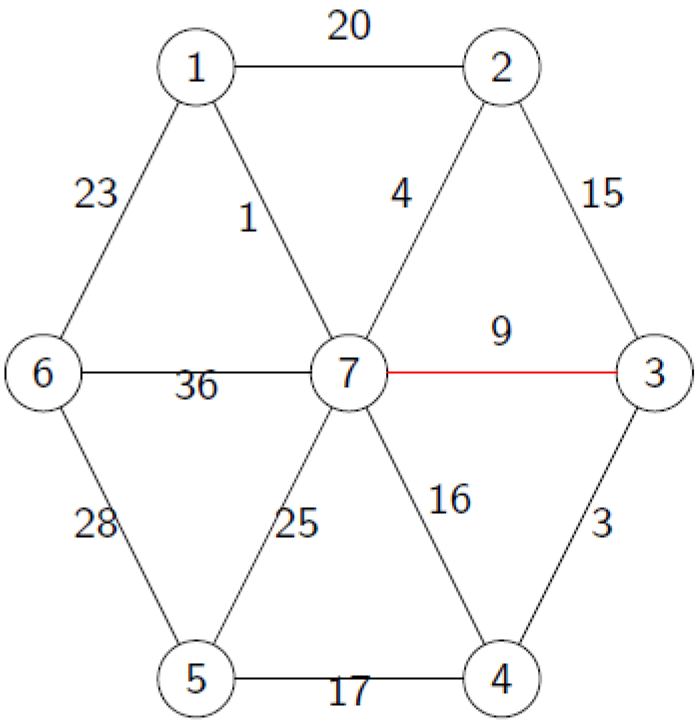


Figure: Graph G

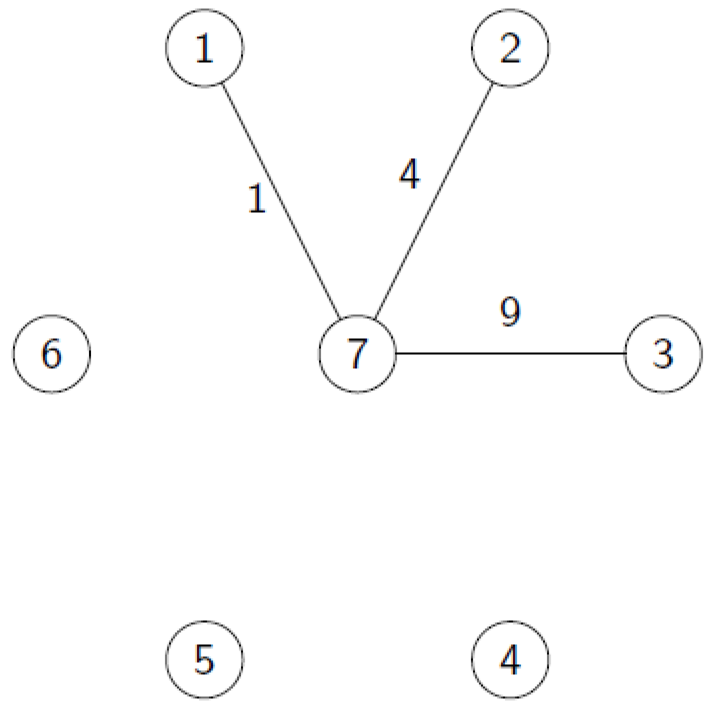


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

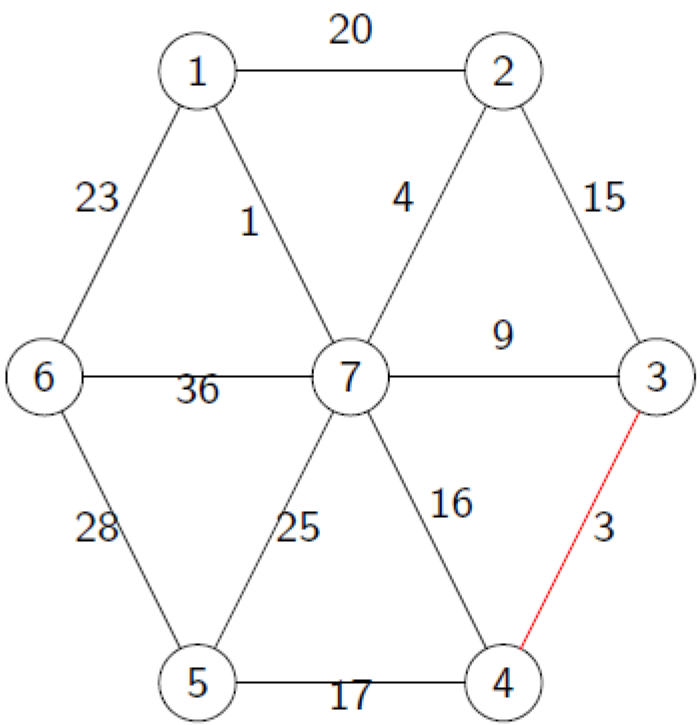


Figure: Graph G

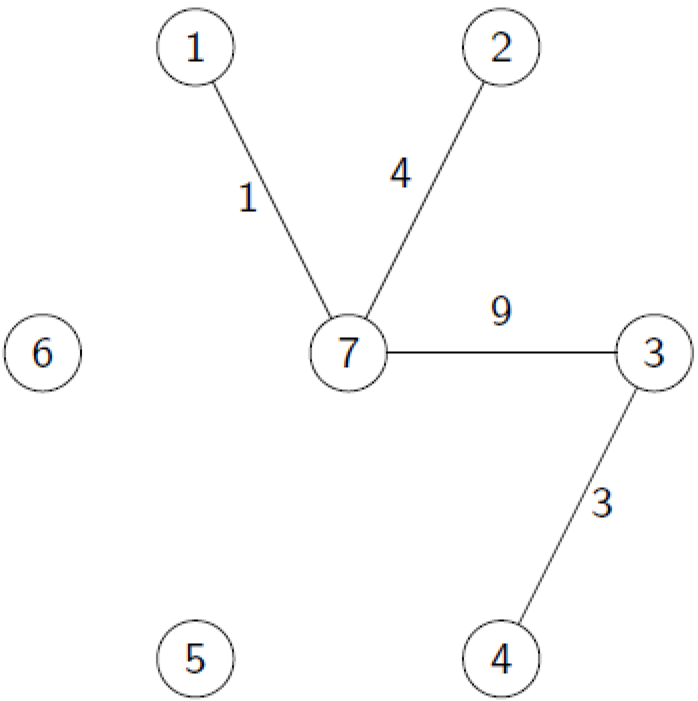


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

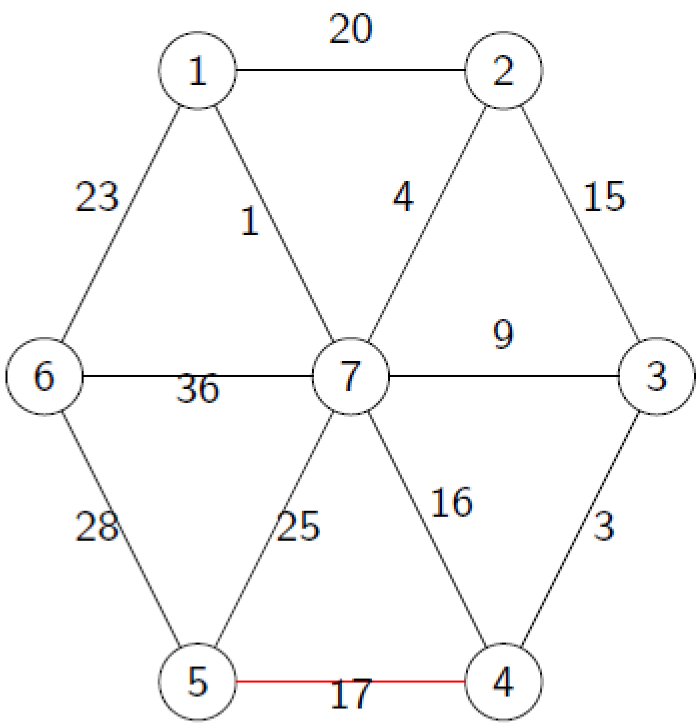


Figure: Graph G

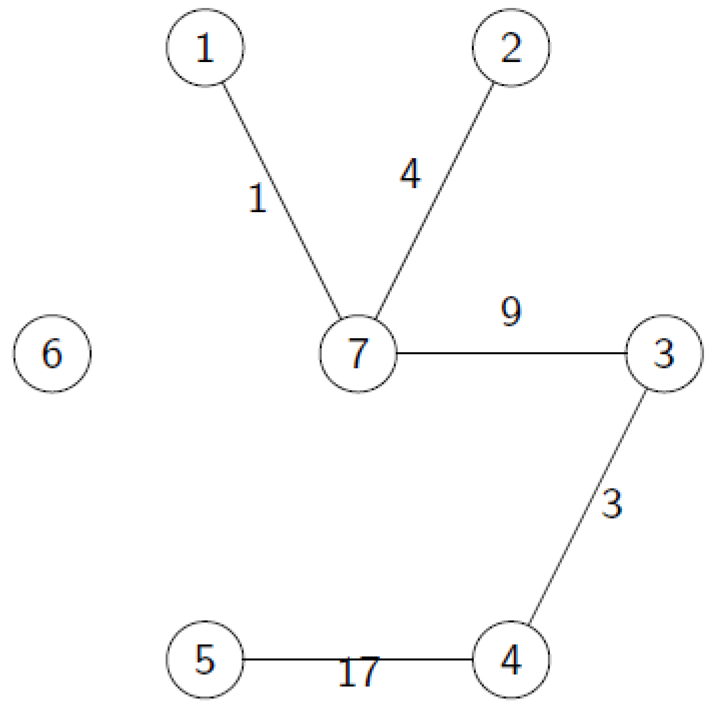


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

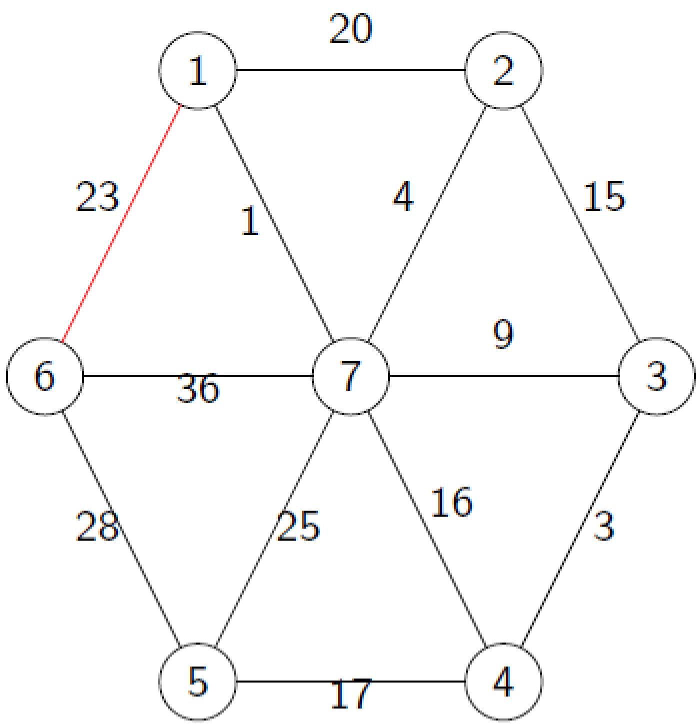


Figure: Graph G

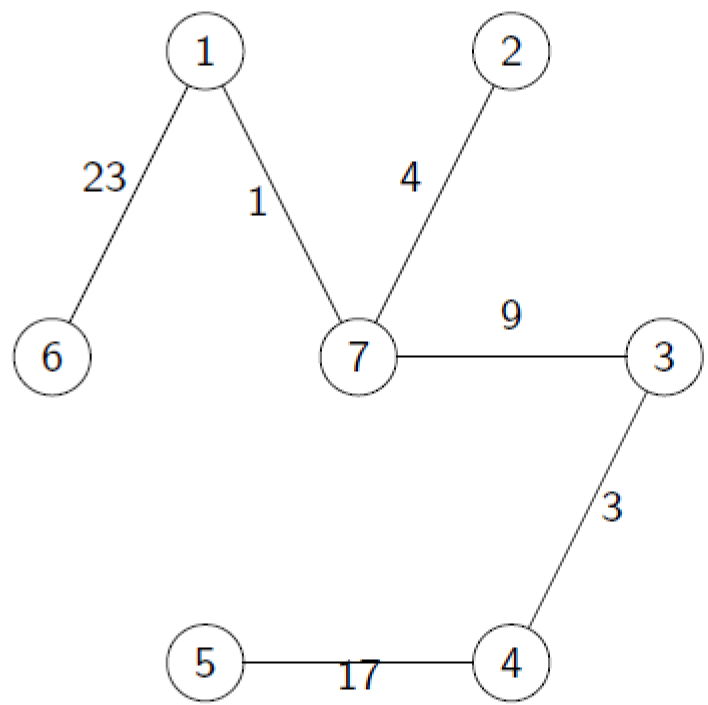


Figure: MST of G

Correctness

- For now, we will assume that all edge costs are distinct
  - As such there is a unique MST

- This can be relaxed later

Cut property

Cut property in plain English:

If we can partition the graph into two parts, and an edge e is the cheapest edge connecting the two parts, then e must be in the MST.

Now let's talk in Math:

Let $S \subset V (\neq \emptyset \text{ and } \neq V)$. Let $e = (v, w)$ be the minimum cost edge with one end in S and the other end in $V \setminus S$. Then e must be in the MST.

Proof Attempt (by contradiction)

Suppose (for contradiction) that e is not in MST T

- Since T is connected (being a spanning tree), there must be some edge f with one end in S and the other in $V \setminus S$
- Since e is cheaper than f, we can replace f with e to get a cheaper spanning tree

- Cheaper? Yes, but can we be sure it will be a spanning tree?
  - When take an edge of a tree and add another one, it might not be a tree anymore

Error in proof:

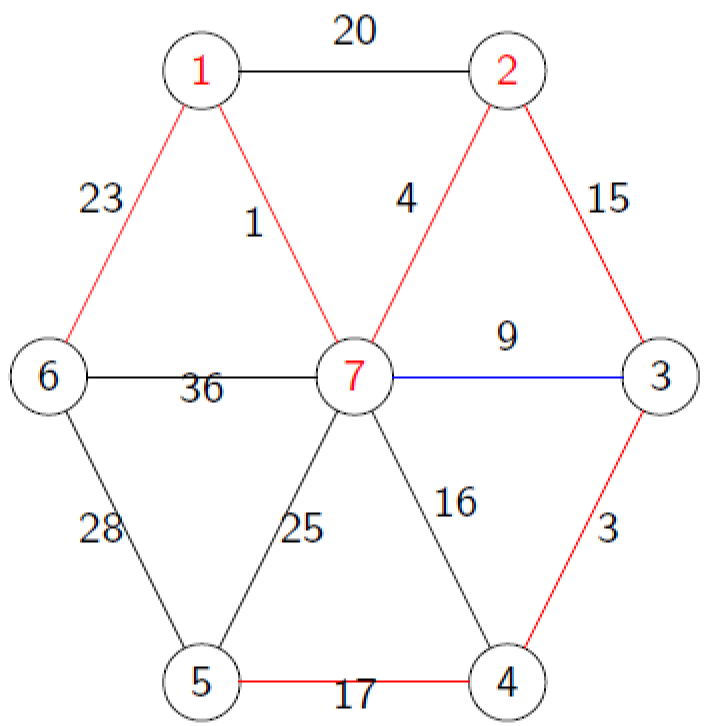
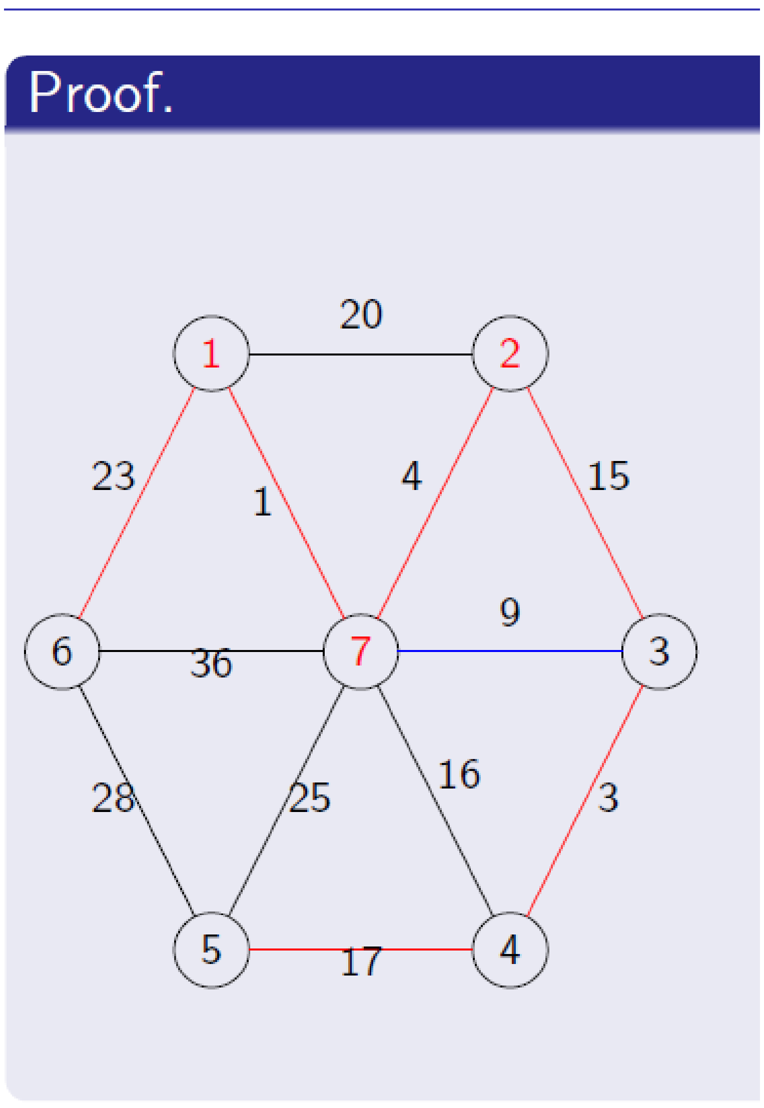


Figure: Problematic example. $S = \{1, 2, 7\}$, $e = (7, 3)$, $f = (1, 6)$

Proof of cut property



- Suppose $e = (v, w)$ is the cheapest edge connecting two parts S and $V - S$ e.g., (7, 3) for {1,2,7} vs. {3,4,5,6} and it's not in the MST T
- Consider T, there must be a path from v to w
  - because T is connected
- On this path, there must be an edge connecting S and V-S
  - In this case it is (2,3)
- If we remove this edge and add e to T, it will be cheaper and will be a tree – so our assumption can't be true, e must be in the MST T

Correctness of Prim's algorithm

- Prim's algorithm: pick the edge with minimum attachment cost to current tree, and add it to current tree.
- Proof of correctness
  – At every iteration, we select the edge e that is the cheapest edge cross the current tree T and the rest of the graph.

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

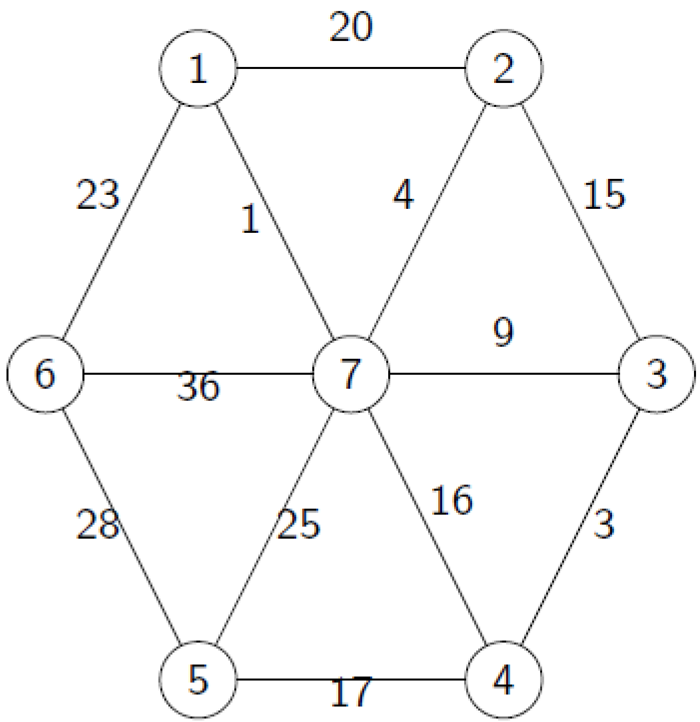


Figure: Graph G

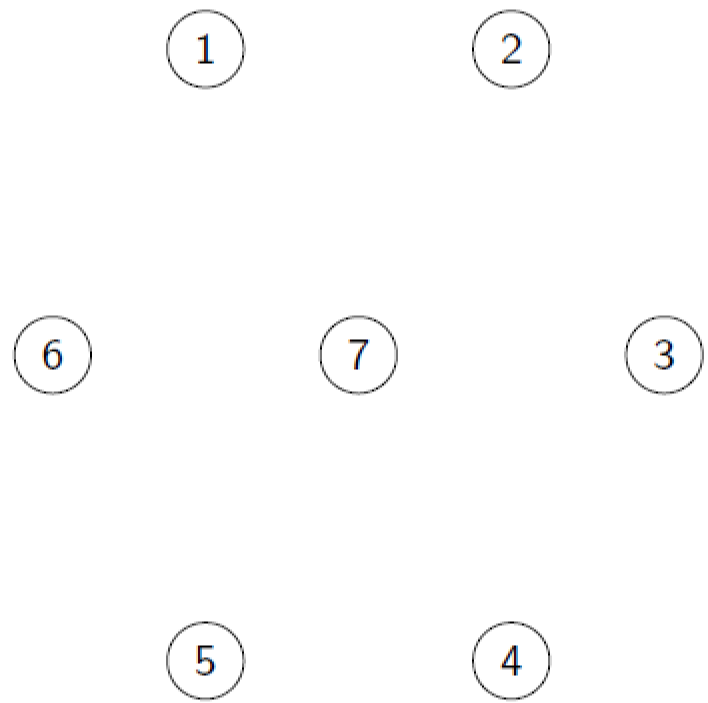


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

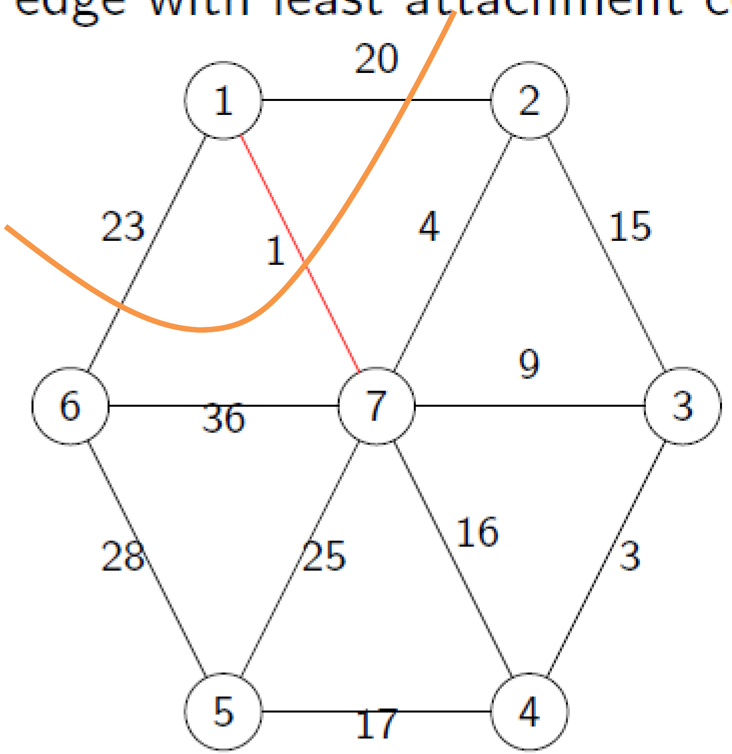


Figure: Graph G

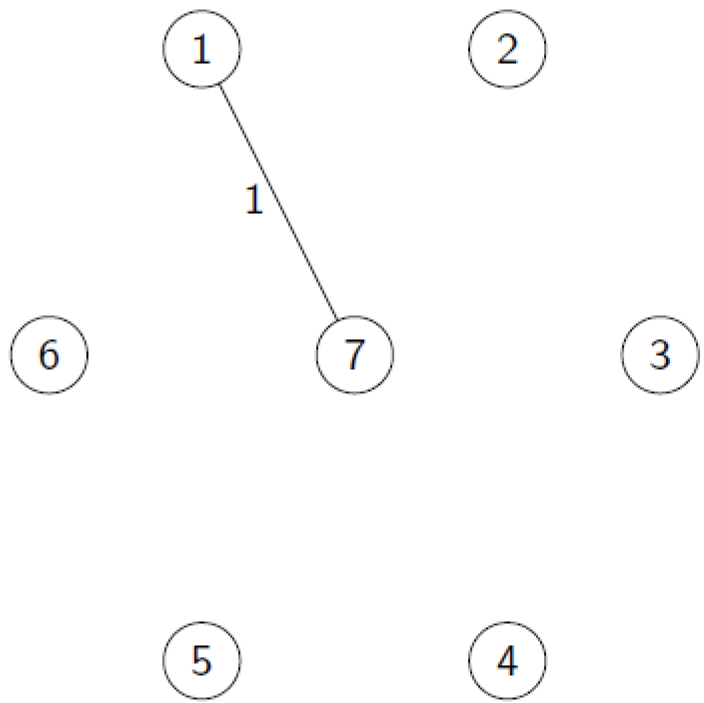


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

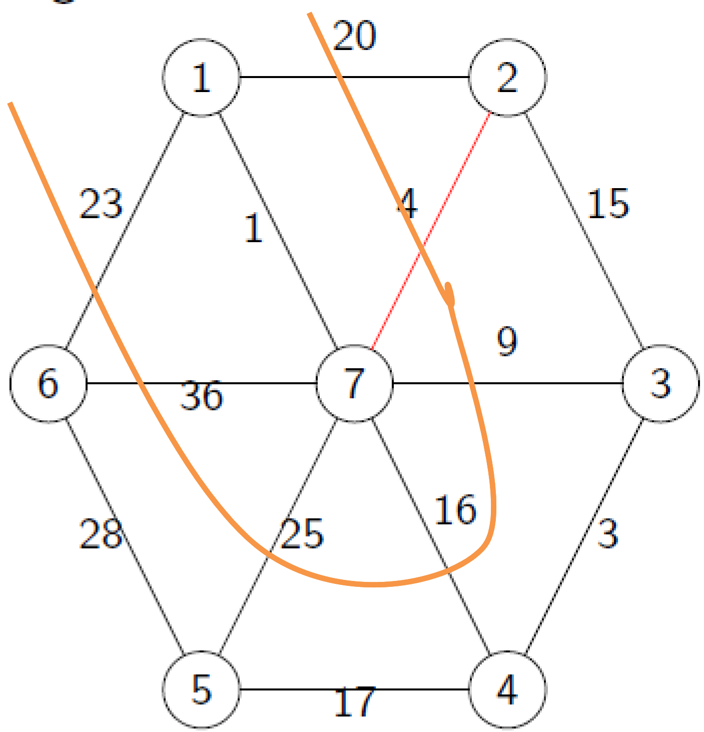


Figure: Graph G

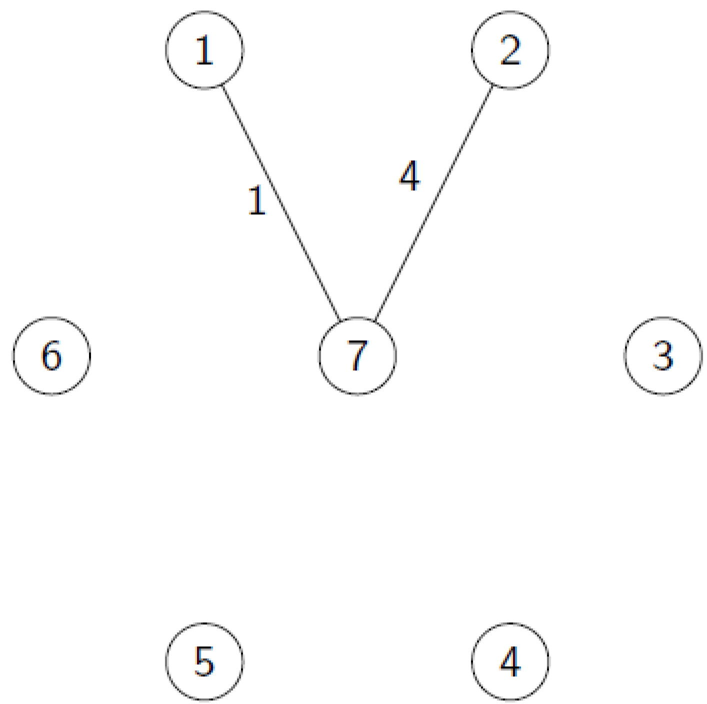


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

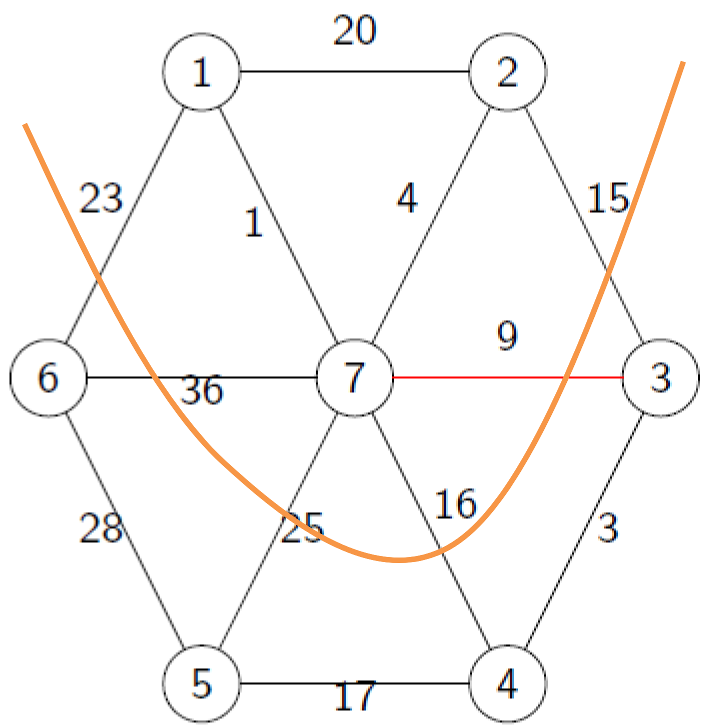


Figure: Graph G

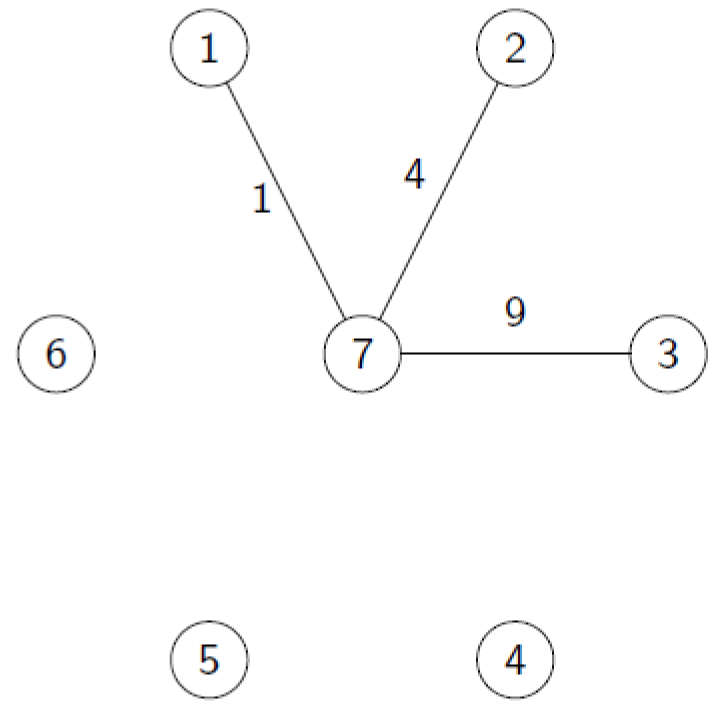


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

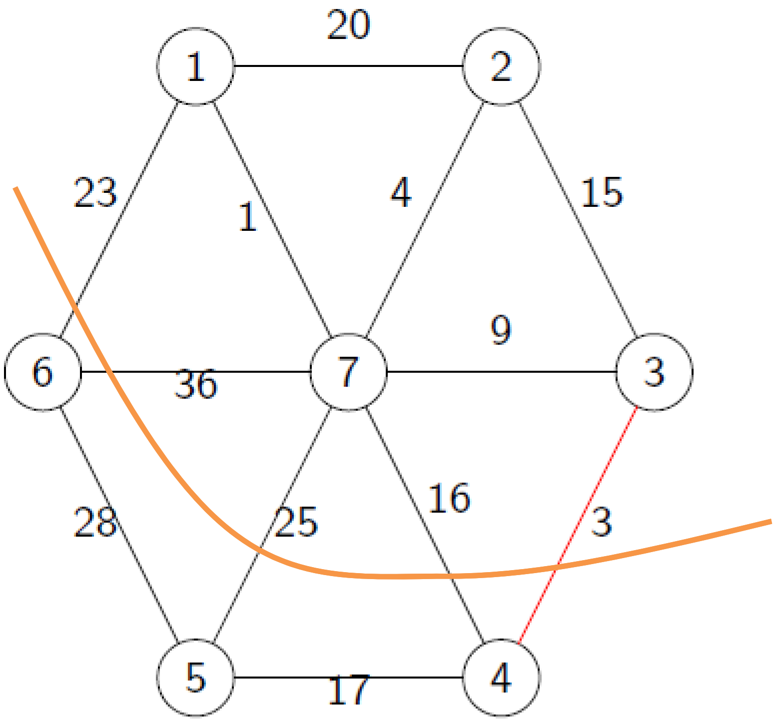


Figure: Graph G

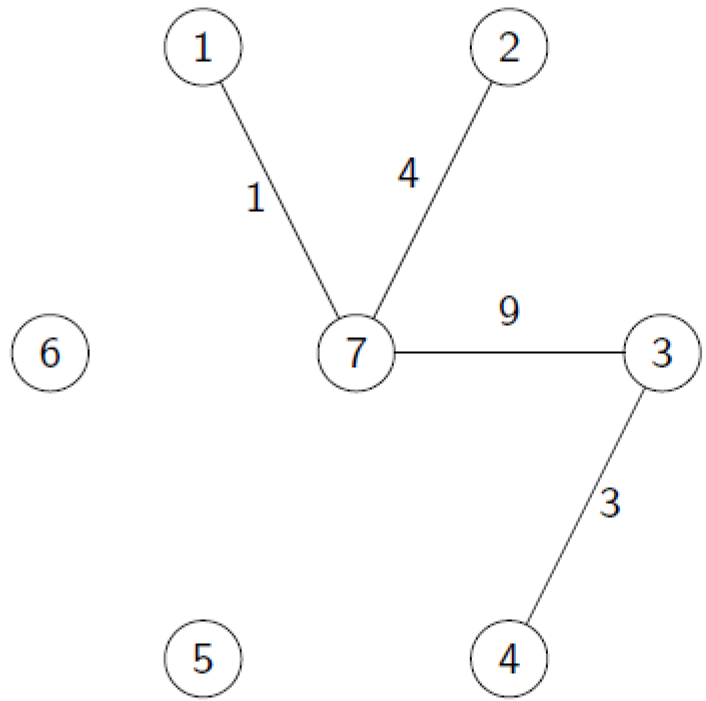


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

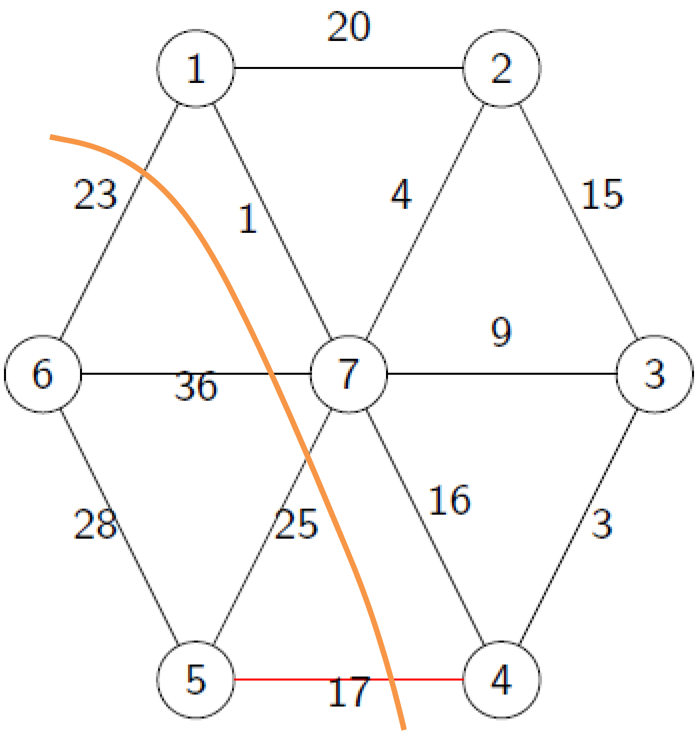


Figure: Graph G

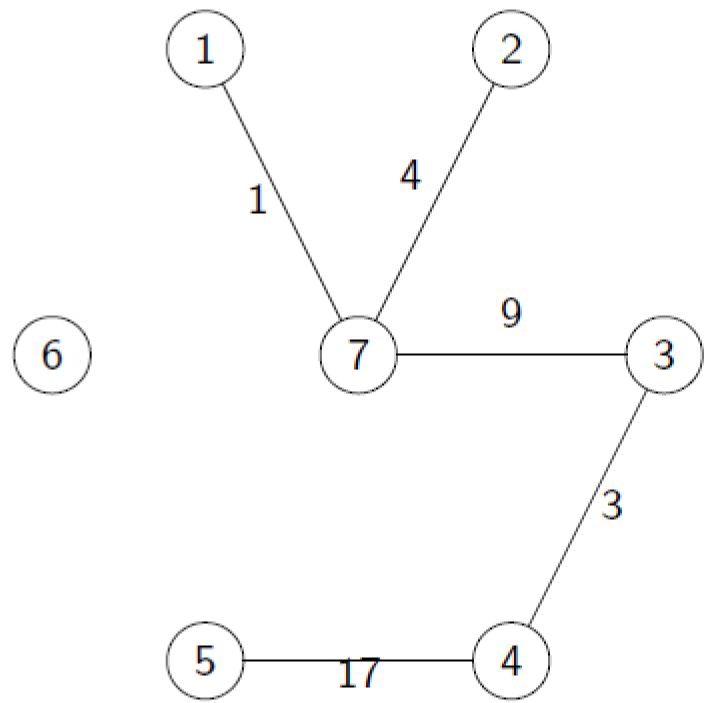


Figure: MST of G

Prim's algorithm

T maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to T.

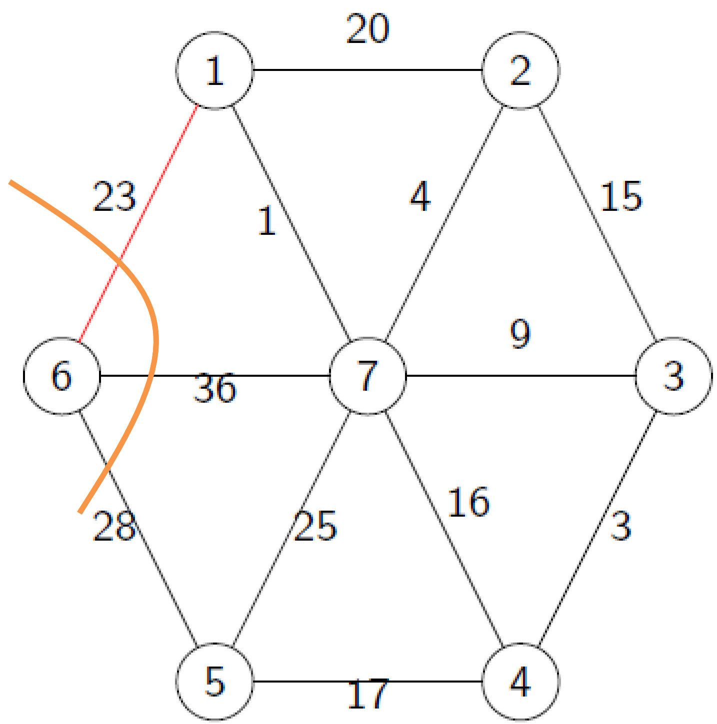


Figure: Graph G

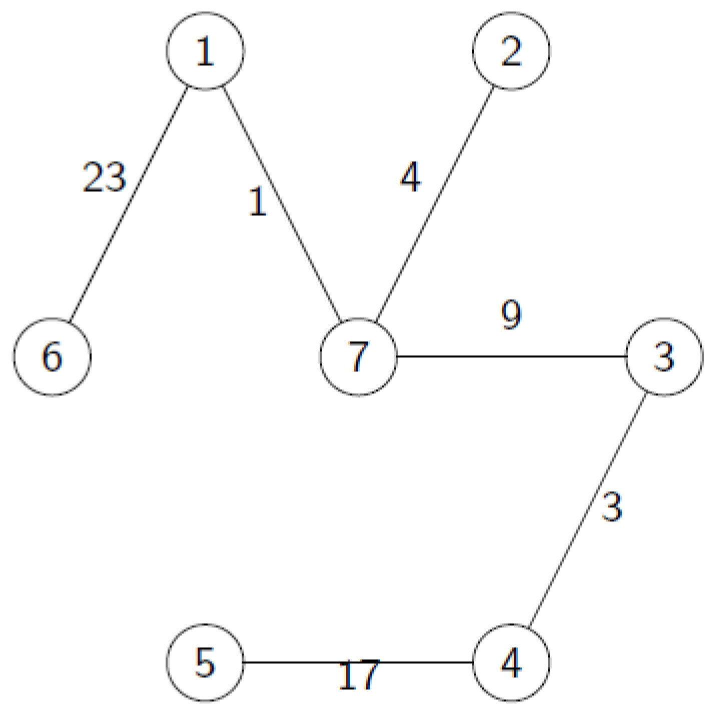


Figure: MST of G

Correctness of Kruskal's algorithm

- Kruskal's algorithm starts with every node in its own connected component
- Each iteration it adds the cheapest edge that links two different connected components
- When an edge $\mathrm{e} = (v, w)$ is selected, let S be the connected component for v (or w, equivalently)
- e must be the cheapest edge connecting S to the rest of the graph

Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

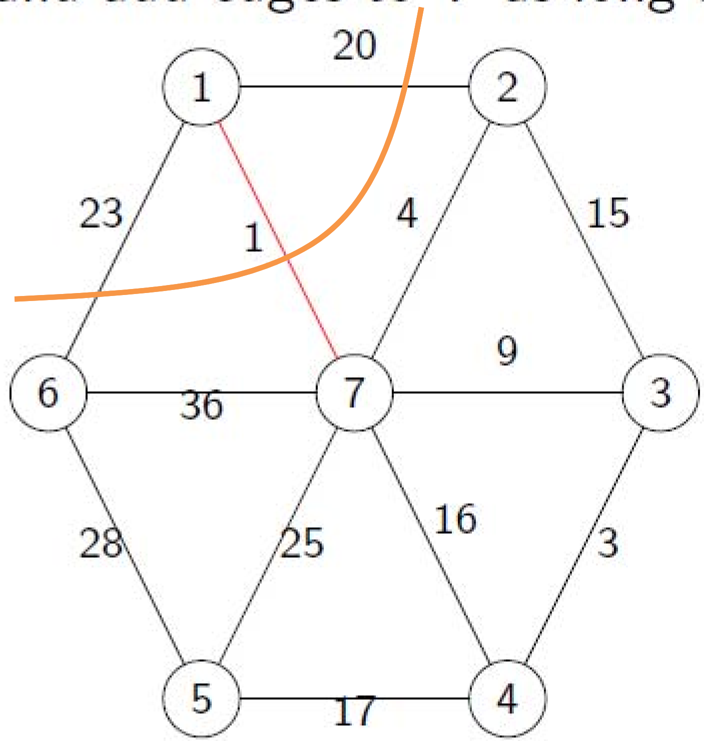


Figure: Graph G

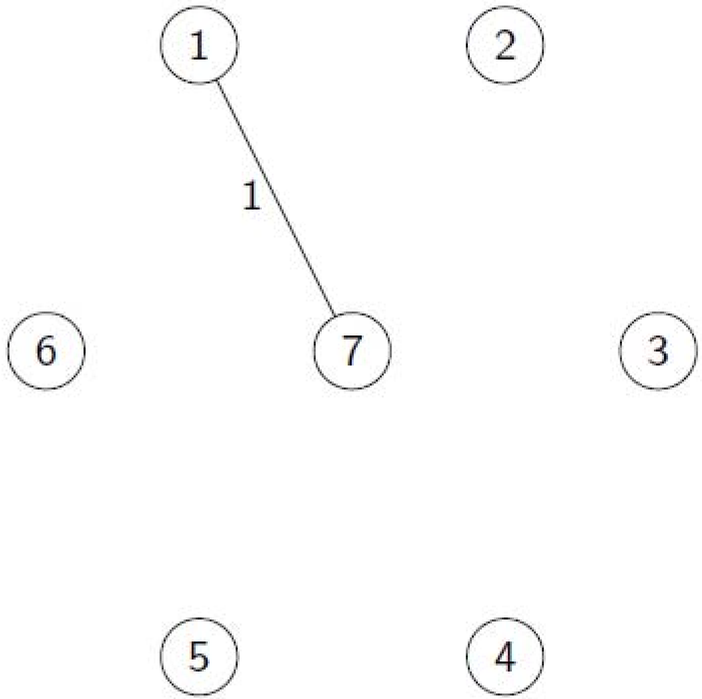


Figure: MST of G

Kruskal's algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

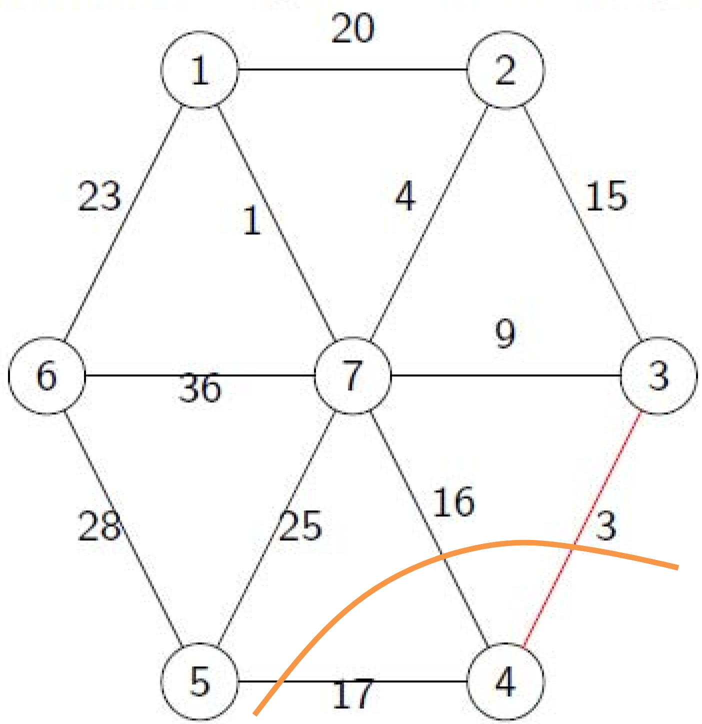


Figure: Graph G

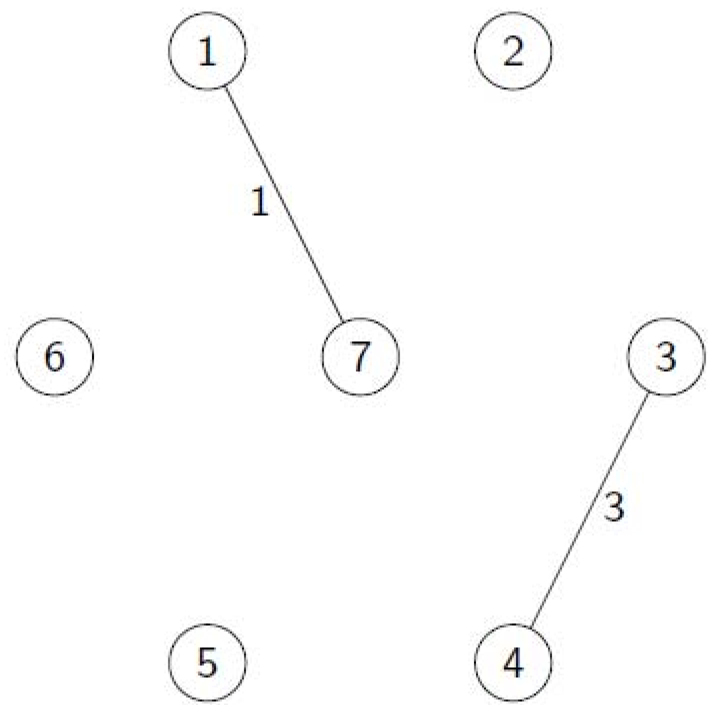


Figure: MST of G

Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

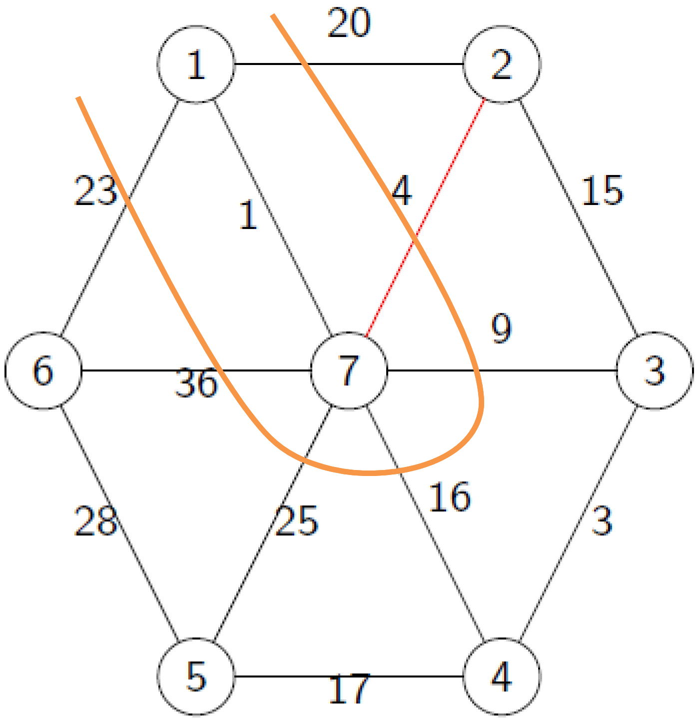


Figure: Graph G

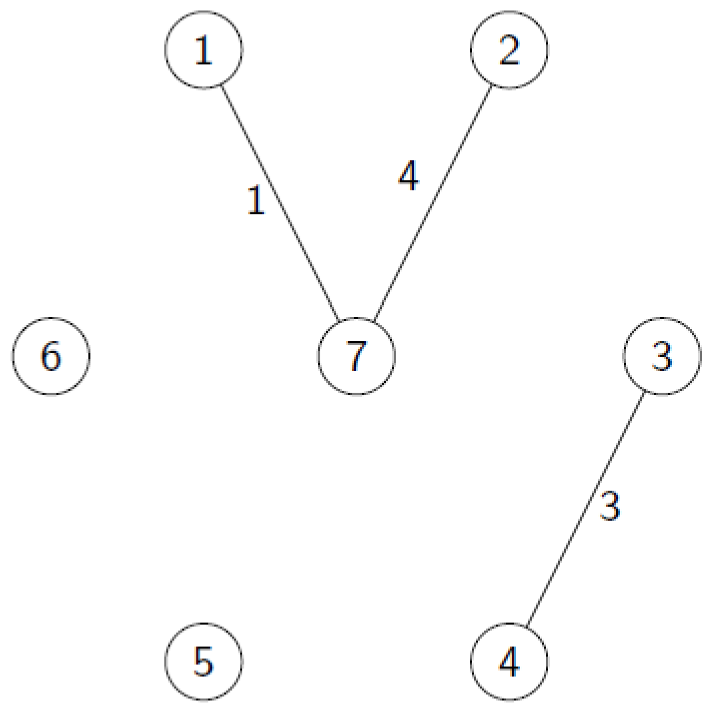


Figure: MST of G

Kruskal's algorithm

Process edges in the order of their costs (starting from the least) and add edges to T as long as they don't form a cycle.

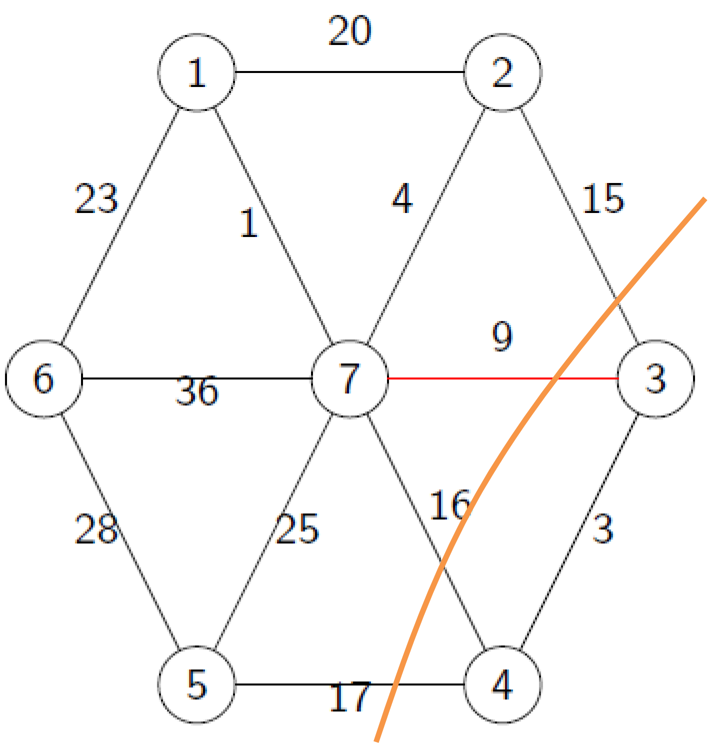


Figure: Graph G

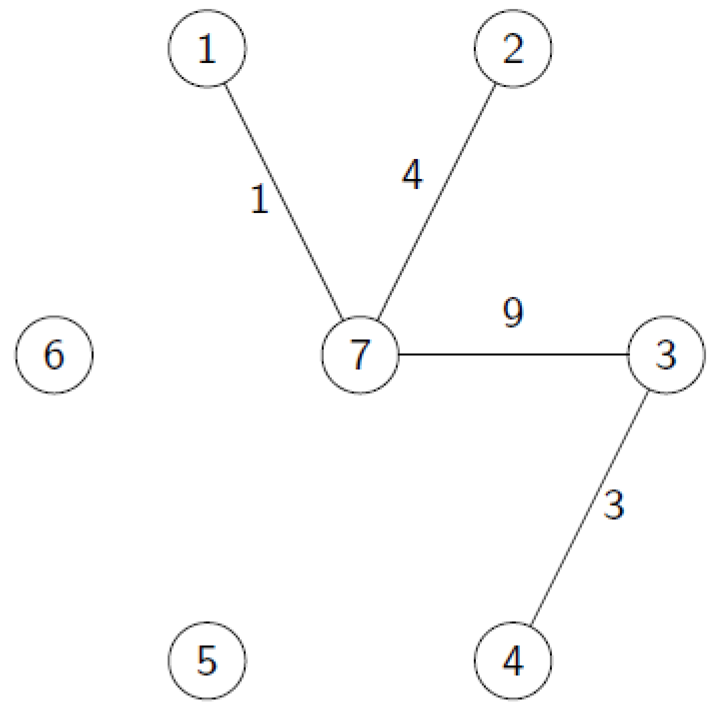


Figure: MST of G

When edge costs are not distinct

- Order edges lexicographically to break ties
  - Based on this tie breaking, we will have one unique MST
- Both Prim's, kruscal's are optimal with respect to the lexicographical ordering

Implementation & Data structure
Briefly …

Prim's algorithm

- Idea: Pick edge with minimum attachment cost to current tree, and add to current tree

Implementing Prim's

E is the set of all edges in G
randomly pick a node $\mathbf{u_0}$ and $S = \{u_0\}$
T is empty (*T stores edge of a MST*)
while $S \neq V$
    pick $e = (v, w) \in E$ such that
        $v \in S, w \notin S$
        e has minimum cost
    $T = T \cup e; S = S \cup w$
return set T

Analysis:

- Number of iterations: $O(|V|)$
- Picking e in each iteration is $O(|E|)$
- Total time $O(|V||E|)$

More efficient implementation

```
procedure prim(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the array prev

for all u ∈ V:
    cost(u) = ∞
    prev(u) = nil
Pick any initial node u_0
cost(u_0) = 0

H = makequeue(V)     (priority queue, using cost-values as keys)
while H is not empty:
    v = deletemin(H)
    for each {v, z} ∈ E:
        if cost(z) > w(v, z):
            cost(z) = w(v, z)
            prev(z) = v
            decreasekey(H, z)
```

$S = S \cup \{v\}, T = T \cup (prev(v), v)$

Maintaining the costs for attaching z to S

Analysis: $(O((|V| + |E|) log\ |V|)$ - assuming using binary heap for priority queue

- each node is inserted and deleted once from the priority queue $(O(|V| \log |V|)$
- Each edge is checked once, leading one possible decreasekey $(O(|E| \log |V|)$

Kruskal's algorithm

- Idea: pick edge of lowest cost and add if it does not form a cycle with existing edges

Implementing Kruskal's

> sort the edges in E in increasing order based on cost
> T is empty (*T stores edge of a MST*)
> for each edge $\boldsymbol{e}$ in sorted order
>   if $T \cup \boldsymbol{e}$ does not creates cycle
> $T = T \cup e$
> return set T

Analysis:

- Sorting the edges: $O(|E| \log |E|)$
- For loop executes $O(|E|)$ times
- Each time, deciding if adding an edge $e = (u, v)$ leads to cycle:
  - Run DFS or BFS on T to see if u and v are connected O(|V|+|E|)
- Total time $O\big(|E| \log|E| + |E|(|V| + |E|)\big) = O(|E|^2)$

Efficient Implementation of Kruskal’s

- Use **Union-by-rank** data structure to maintain disjoint sets
- Each set contains the nodes of a particular connected component
- Initially each node is in a component by itself

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
    makeset(u)        ← Place every node in its own connected component. O(|V|)

X = {}
Sort the edges E by weight        ← |E|log|E|
for all edges {u, v} ∈ E, in increasing order of weight:
    if find(u) ≠ find(v):        ← If u and v are not in the same connected component  O(log |V|)
        add edge {u, v} to X
        union(u, v)        ← Merge the two connected components of u and v. O(C)
```

- Total running time:

$$O|V| + O(|E|\log|E|) + O(|E|\log|V|)$$