20.3
The Algorithms for computing MST

Greedy Template

```
Initially E is the set of all edges in G
T is empty (* T will store edges of a MST *)
while E is not empty do
    choose e ∈ E
    remove e from E
    if (e satisfies condition)
        add e to T
return the set T
```

Main Task: In what order should edges be processed? When should we add edge to spanning tree?

KA PA RD

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.

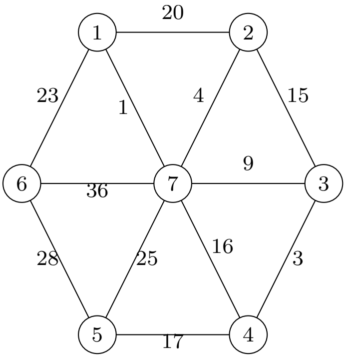


Figure: Graph G

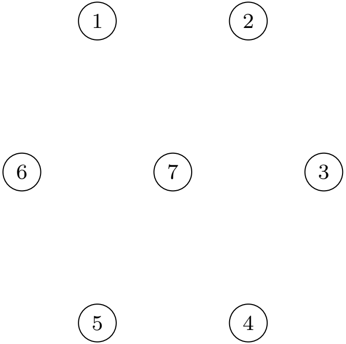


Figure: MST of G

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.

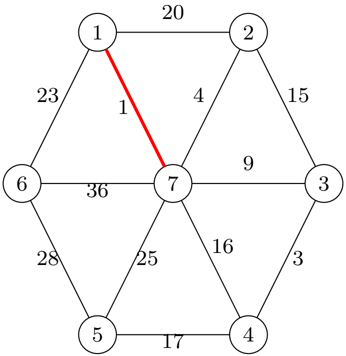


Figure: Graph ***G***

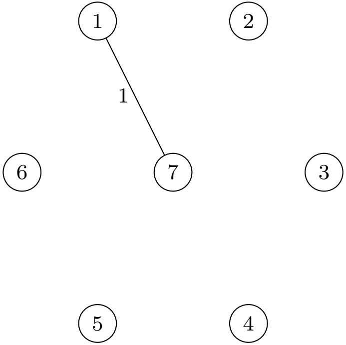


Figure: MST of ***G***

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.

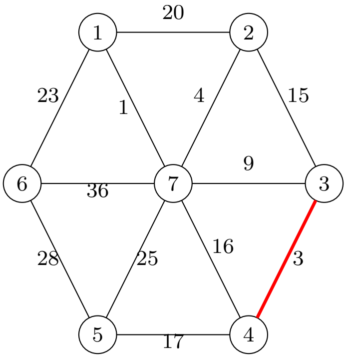


Figure: Graph G

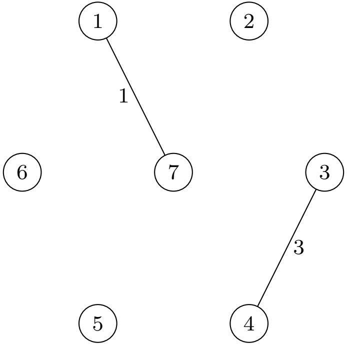


Figure: MST of G

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.

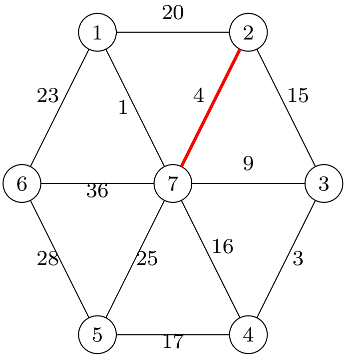


Figure: Graph ***G***

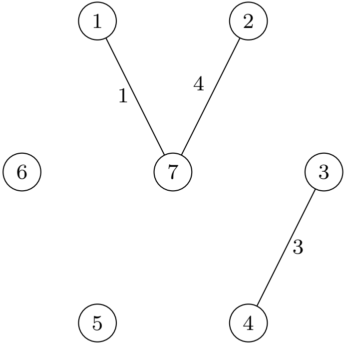


Figure: MST of ***G***

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.

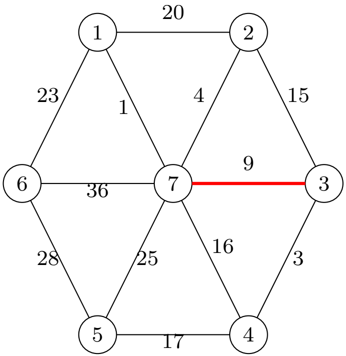


Figure: Graph G

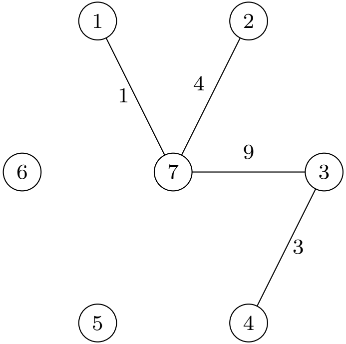


Figure: MST of G

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.

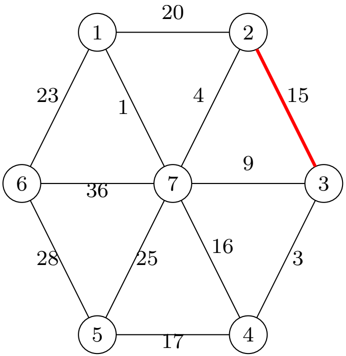


Figure: Graph G

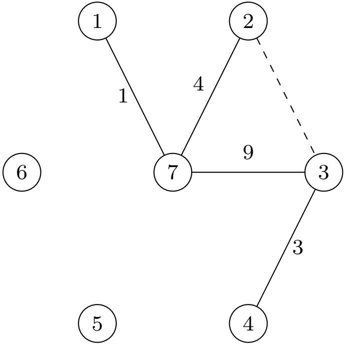


Figure: MST of G

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.

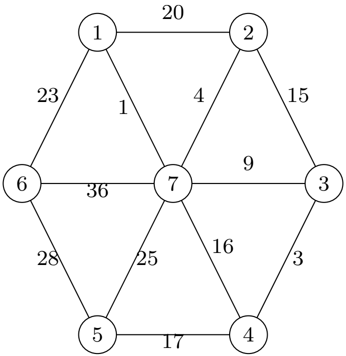


Figure: Graph G

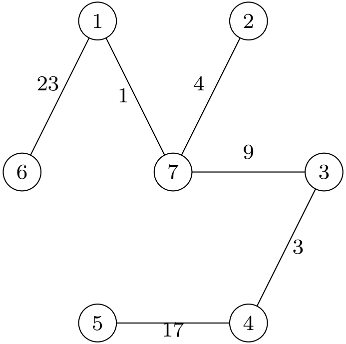
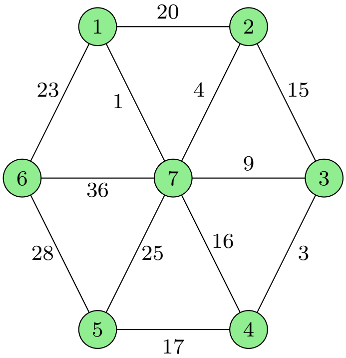
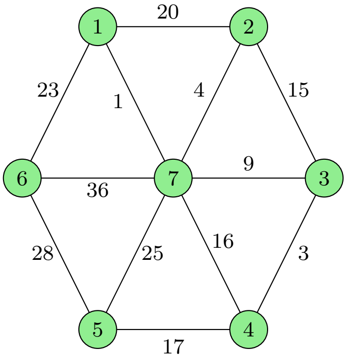


Figure: MST of G

Prim's Algorithm: Animation

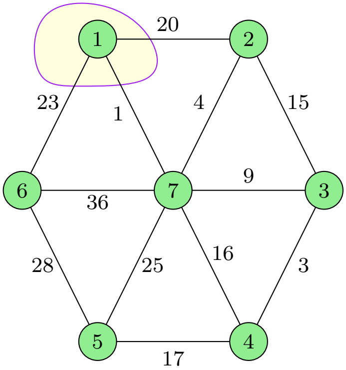
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

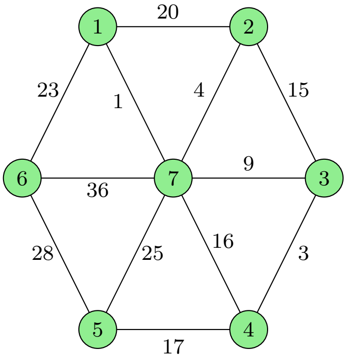




Prim's Algorithm: Animation

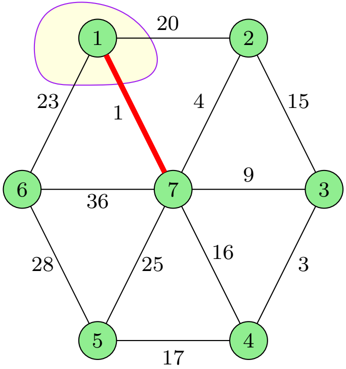
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

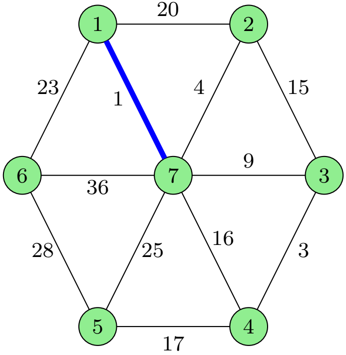




Prim's Algorithm: Animation

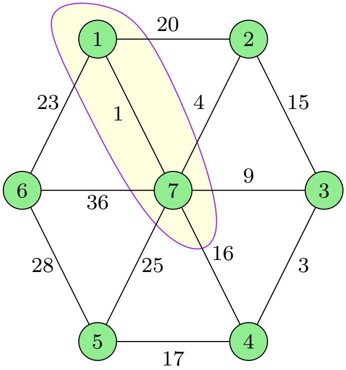
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

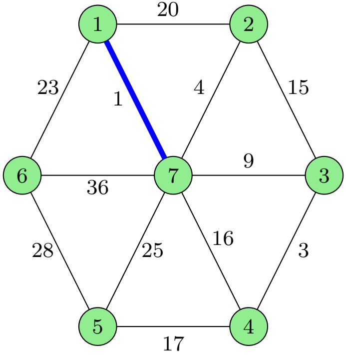




Prim's Algorithm: Animation

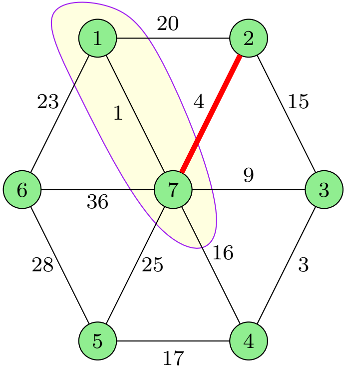
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

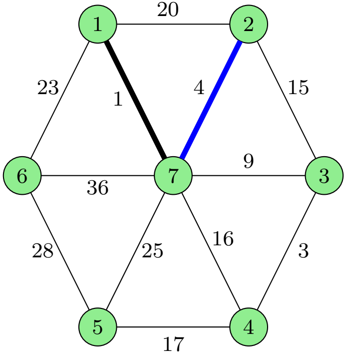




Prim's Algorithm: Animation

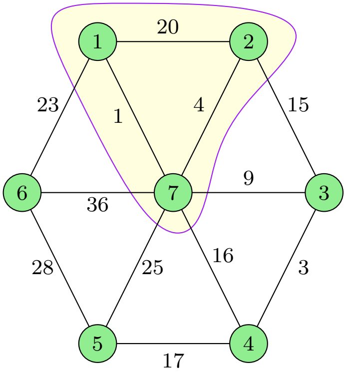
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

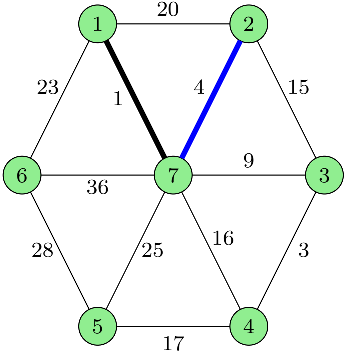




Prim's Algorithm: Animation

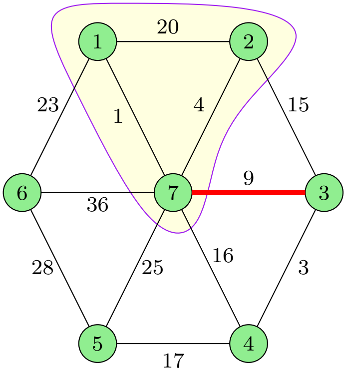
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

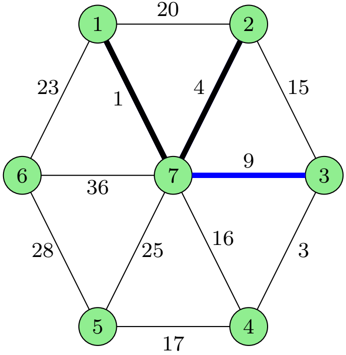




Prim's Algorithm: Animation

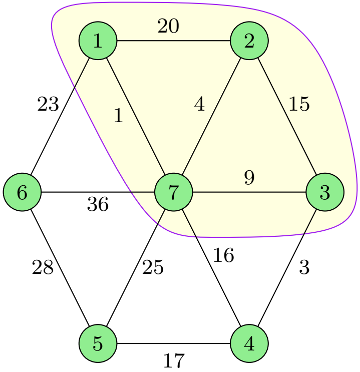
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

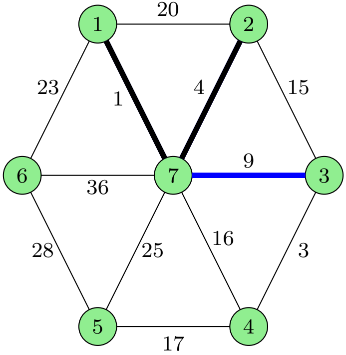




Prim's Algorithm: Animation

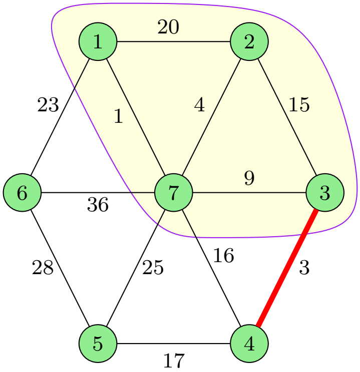
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

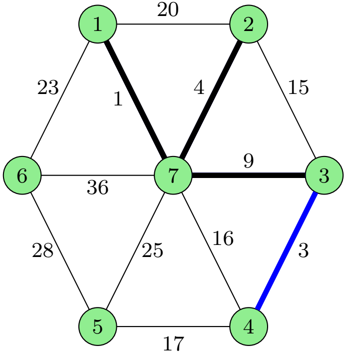




Prim's Algorithm: Animation

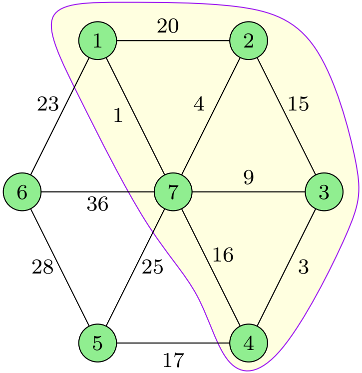
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

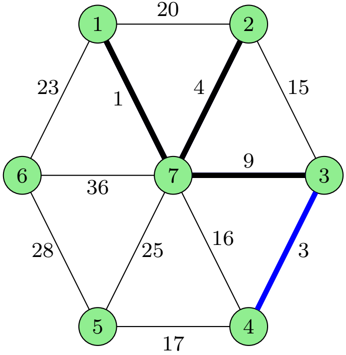




Prim's Algorithm: Animation

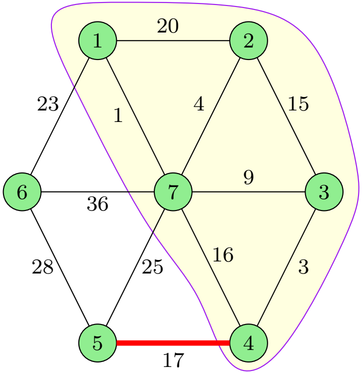
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

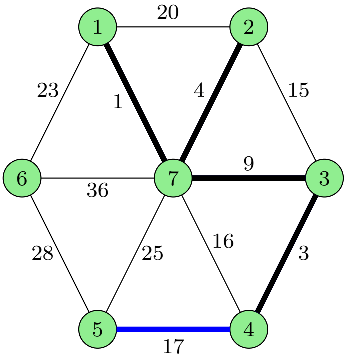




Prim's Algorithm: Animation

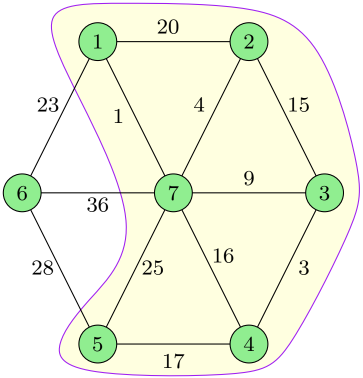
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

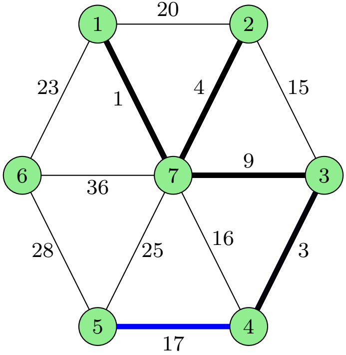




Prim's Algorithm: Animation

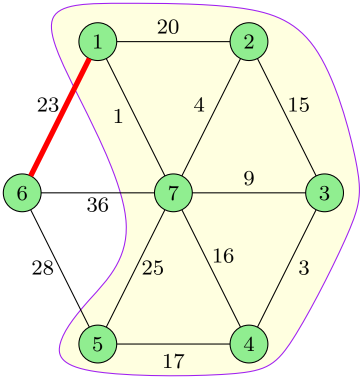
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

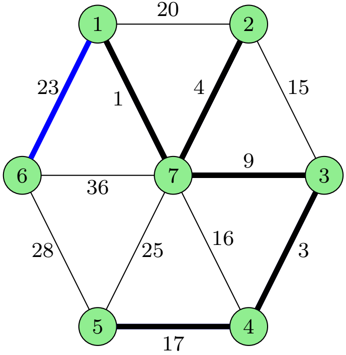




Prim's Algorithm: Animation

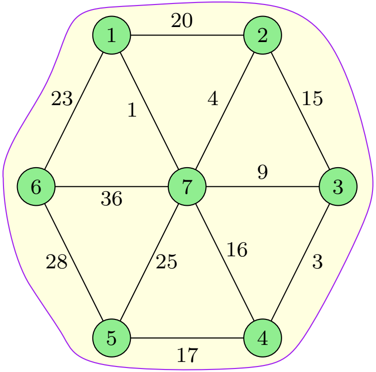
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

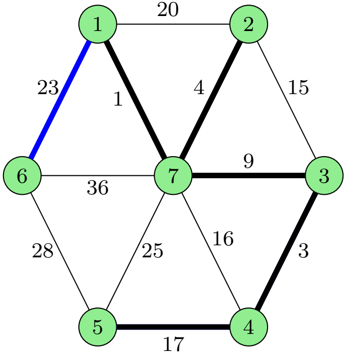




Prim's Algorithm: Animation

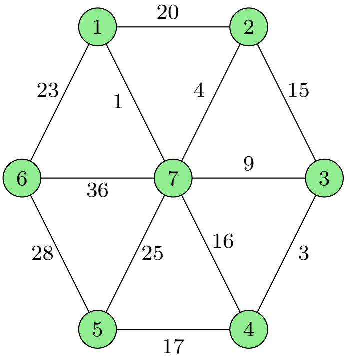
T maintained by algorithm will be a tree. Start with a node in ***T***. In each iteration, pick edge with least attachment cost to ***T***.

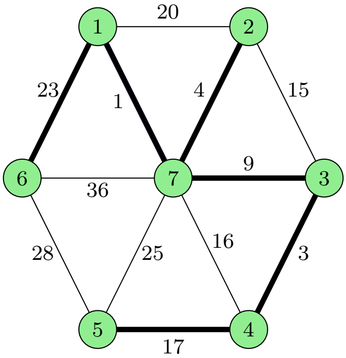




Prim's Algorithm: Animation

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





Reverse Delete Algorithm

```
Initially Z is the set of all edges in G
T ⇐ Z (* T will store edges of a MST *)
while Z is not empty do
    choose e ∈ Z of largest cost
    remove e from Z
    if removing e does not disconnect T then
        remove e from T
return the set T
```

Returns a minimum spanning tree.

Back

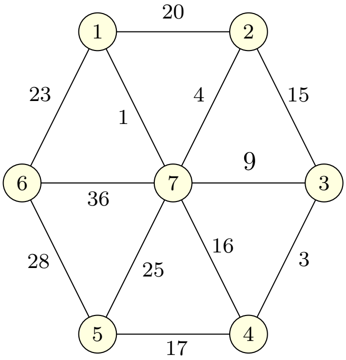
Borůvka's Algorithm

Simplest to implement. See notes.

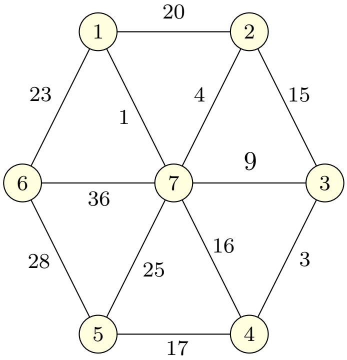
Assume G is a connected graph.

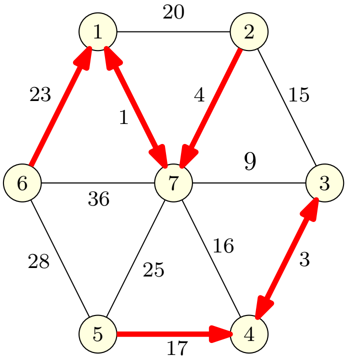
```
T is ∅ (* T will store edges of a MST *)
while T is not spanning do
    X ← ∅
    for each connected component S of T do
        add to X the cheapest edge between S and V \ S
    Add edges in X to T
return the set T
```

Borůvka's Algorithm

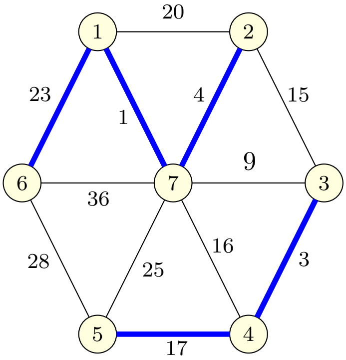


Borůvka's Algorithm

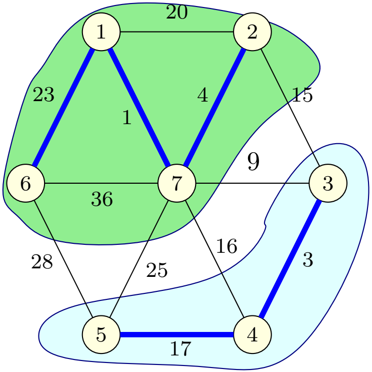




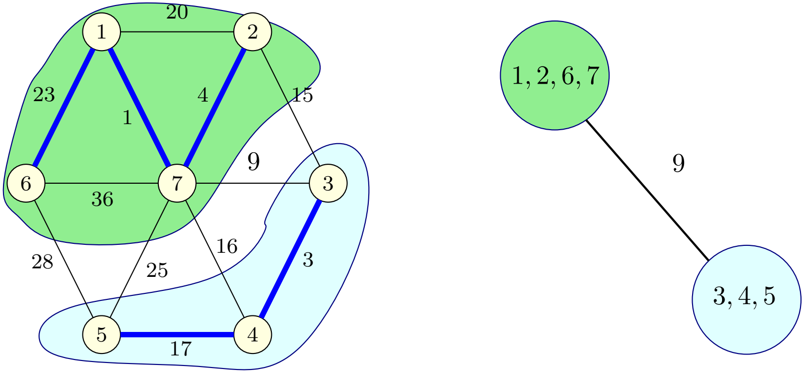
Borůvka's Algorithm



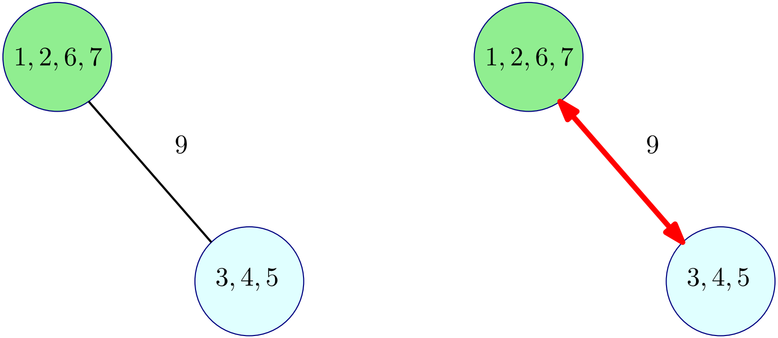
Borůvka's Algorithm



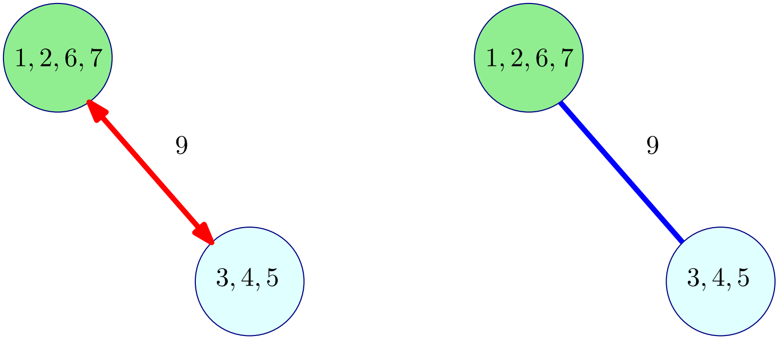
Borůvka's Algorithm



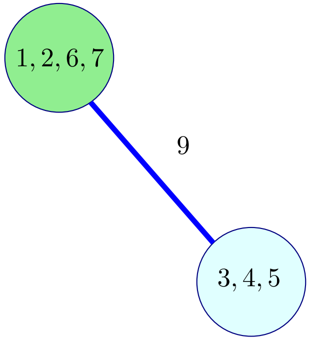
Borůvka's Algorithm



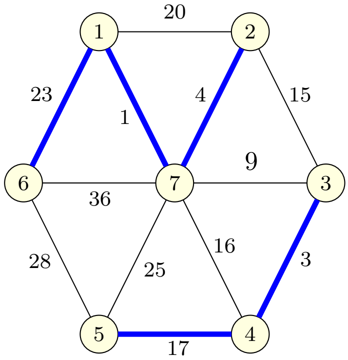
Borůvka's Algorithm



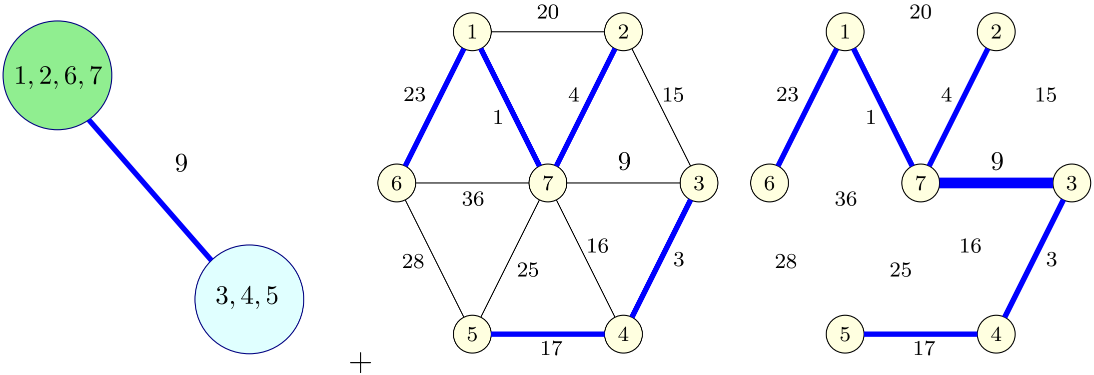
Borůvka's Algorithm



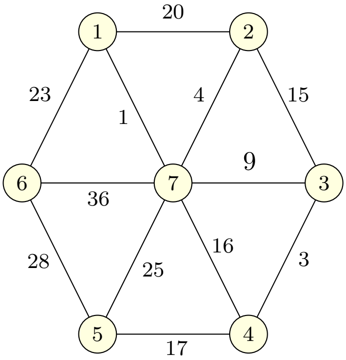
+

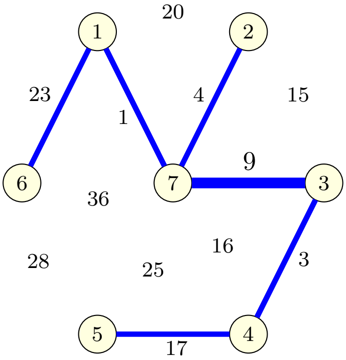


Borůvka's Algorithm



Borůvka's Algorithm





THE END

...

(for now)