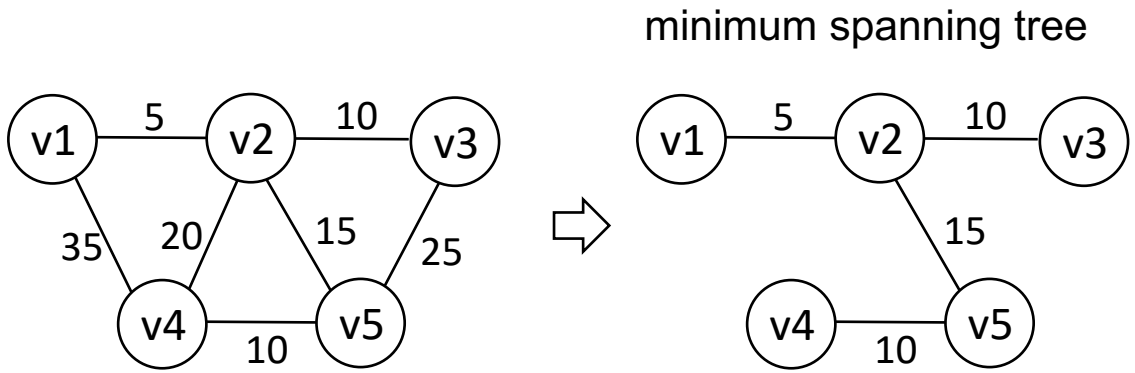
DS:
Minimum Spanning Tree

Liwei

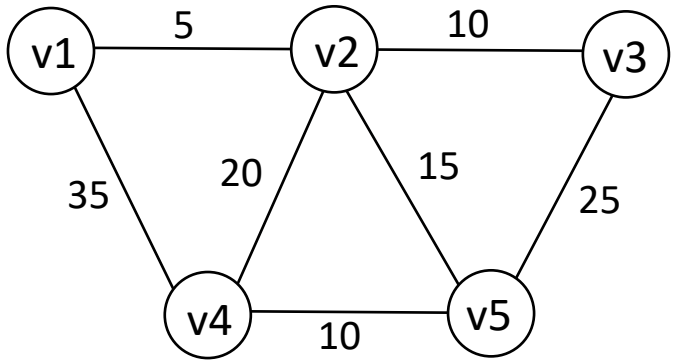
What is Minimum Spanning Tree (MST)?

- A minimum spanning tree is a subset of the edges of a connected, undirected graph that
  - connects all the vertices together
  - without any cycles
  - with minimum total edge weight.

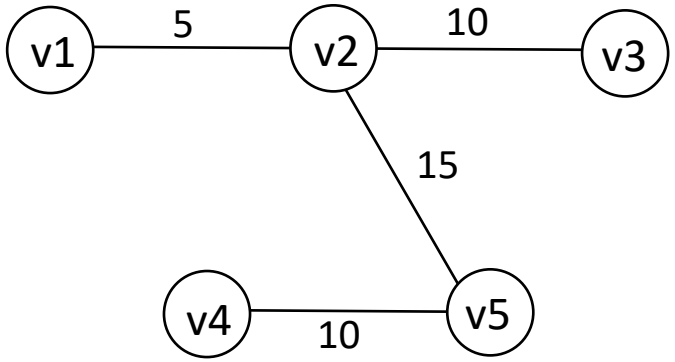


- Example:
- Lets say we want to connect 5 islands using bridges
- Cost of each bridge varies depending on different factors
- How to determine which bridge to be constructed and which not?

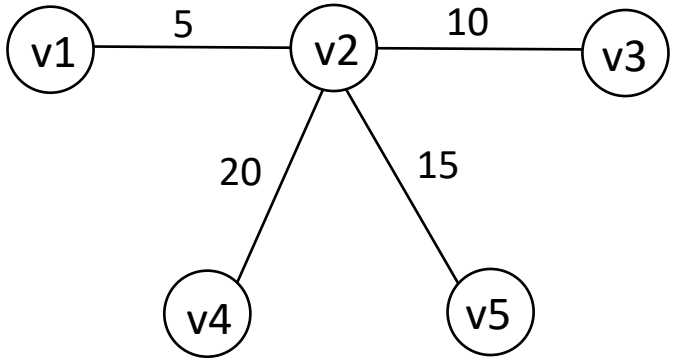
MST vs. Single source shortest path



- Minimum spanning tree



- Single source shortest path



Disjoint Set

- What is Disjoint Set?
- It is a Data Structure that keeps track of sets of elements which are partitioned into a number of disjoint and non-overlapping sets.
- It is used in various algorithms. One of which is Kruskal algorithm to detect cycle in undirected graph.

35

Standard operations of Disjoint set:

- MakeSet(N)
  - create n sets for N elements, used only once to create initial set
- Union(x,y)
  - used to merge both the sets
- FindSet(x)
  - return the representative set in which this element exists

MakeSet(N) → {A}, {B}, {C}, {D}, {E}

Union(A, B) → {AB}, {C}, {D}, {E}

Union(A, E) → {ABE}, {C}, {D}

FindSet(A) → {ABE}

FindSet(B) → {ABE}

FindSet(C) → {C}

Algorithm – union (s1, s2)

```
union (s1, s2)
        if s1 and s2 are in same set, then return
        else
                if s1 is bigger, then merge s2 into s1
                else merge s1 into s2
        return merged set
```

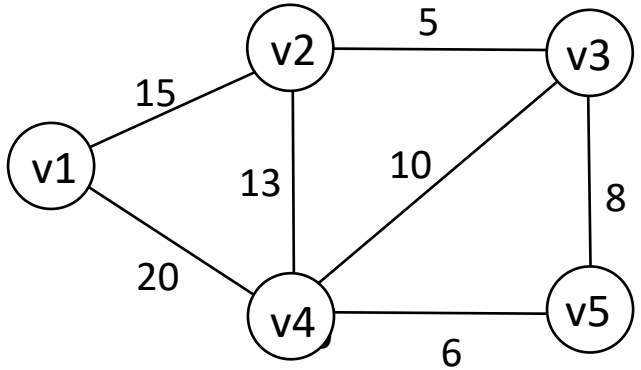
O(n)

Time complexity: O(n)

Kruskal Algorithm

Kruskal Algorithm

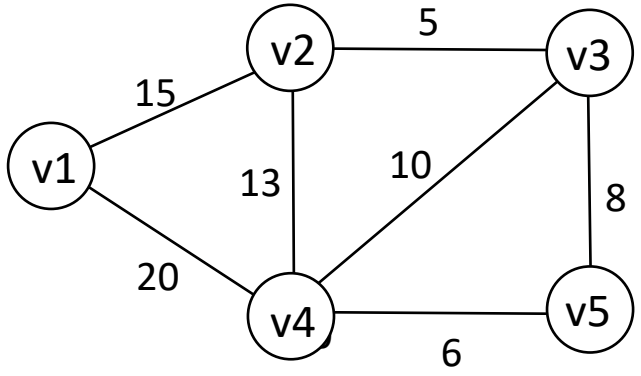
- Kruskal's algorithm is a greedy algorithm
- It finds a minimum spanning tree for a connected weighted graph by
  - Adding increasing cost arcs at each step
  - Also avoiding cycle in every step.

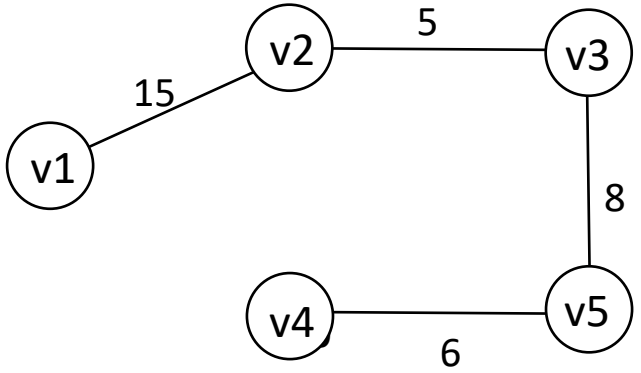


35

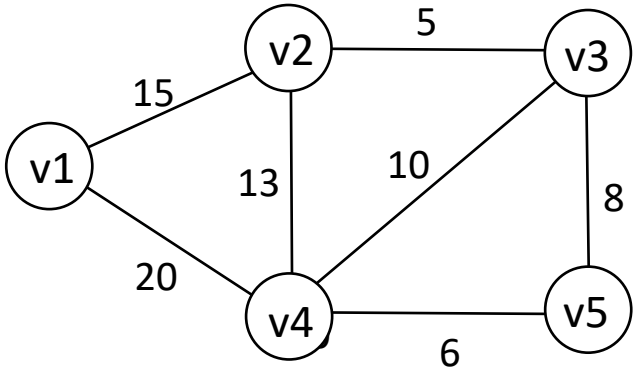
Kruskal Algorithm

- Kruskal's algorithm is a greedy algorithm
- It finds a minimum spanning tree for a connected weighted graph by
  - Adding increasing cost arcs at each step
  - Also avoiding cycle in every step.





Kruskal Algorithm




```
MST-krustal(G)
        for each vertex: Makeset(X)
        sort each edge in an increasing order by weight
        for each edge(u,v) do:
                if findSet(u) != findSet(v)
                        Union(u,v)
                                cost = cost + edge(u,v)
```

Kruskal Algorithm

```
MST-krustal(G)
        for each vertex: Makeset(X)                          O(v)
        sort each edge in an increasing order by weight      O(e log(e))
        for each edge(u,v) do:                               O(e)
                if findSet(u) != findSet(v)                  O(1)
                        Union(u,v)                           O(v)
                                cost = cost + edge(u,v)      O(1)
```

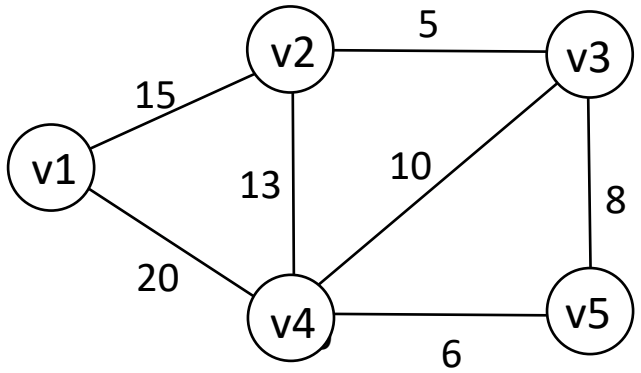
O(e,v)

Time complexity: O(V+ElogE + EV)
= O(E logE)

Prim’s Algorithm

Prim's Algorithm

- Prim's algorithm is also a greedy algorithm
- Process
  1. Take any vertex as Source and mark weight of all the vertex as infinite and source as 0
  2. For every adjacent unvisited vertex of current vertex (e.g., the minimum weighted vertex)
  3. If current weight of this 'adjacent vertex' is more than current edge, then update adjacent vertex's weight and parent.
  4. Mark current vertex as visited
  5. Do above steps for all the vertices in increasing order of weights



Prim Algorithm

```
MST-prims (G)
        create a PriorityQueue(Q)
        insert all the vertices into Q such that key value of starting vertex is 0 and others is infinite.
        while Q is not empty
                currentVertex = dequeue(Q)
                for every adjacent unvisited vertex of currentVertex
                        if current weight of this adjacent vertex is more than current edge
                                then update adjacent vertex's distance and parent
                mark currentVertex as visited
        print all the vertices with weights
```

Prim Algorithm

```
MST-prims (G)
        create a PriorityQueue(Q)                                   O(1)
        insert all the vertices into Q such that key value of starting vertex is 0 and others is infinite.   O(V)
        while Q is not empty                                        O(V)
                currentVertex = dequeue(Q)                          O(log V)
                for every adjacent unvisited vertex of currentVertex   O(V)
                        if current weight of this adjacent vertex is more than current edge   O(1)
                                then update adjacent vertex's distance and parent   O(log V)   } O(log V)
                mark currentVertex as visited                       O(1)
        print all the vertices with weights                         O(V)
```

O(V logV) } O(V logV)

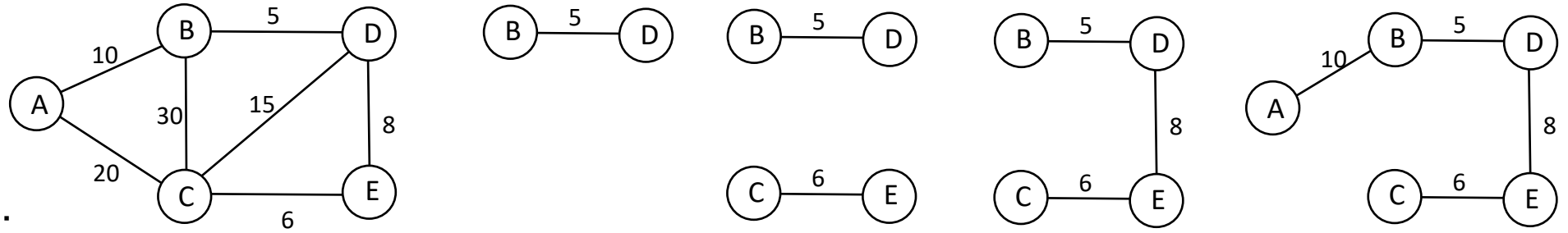
O(V² logV)

Time complexity = O(1)+O(V)+O(V²logV)
= O(V²logV) = O(E logV)

Time complexity: O(E logV)

Kruskal vs. Prim's Algorithms

- Kruskal:
  - Algorithm's concentration is on edges
  - We finalize edge in every iteration



- Prim's:
  - Algorithm's concentration is on vertices
  - We finalize vertex in every iteration