Graph Algorithms

CptS 223 – Advanced Data Structures

Larry Holder
School of Electrical Engineering and Computer Science
Washington State University

Minimum Spanning Trees

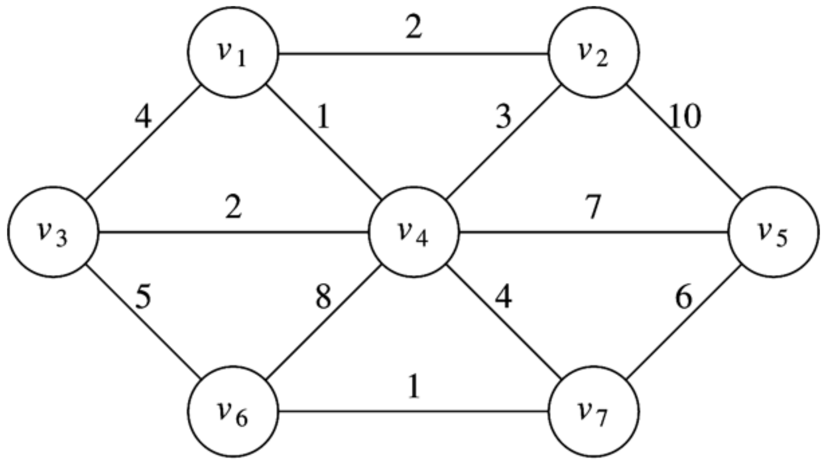
- Find a minimum-cost set of edges that connect all vertices of a graph
- Applications
  - Connecting "nodes" with a minimum of "wire"
    - Networking
    - Circuit design
  - Collecting nearby nodes
    - Clustering, taxonomy construction
  - Approximating graphs
    - Most graph algorithms are faster on trees

Minimum Spanning Tree

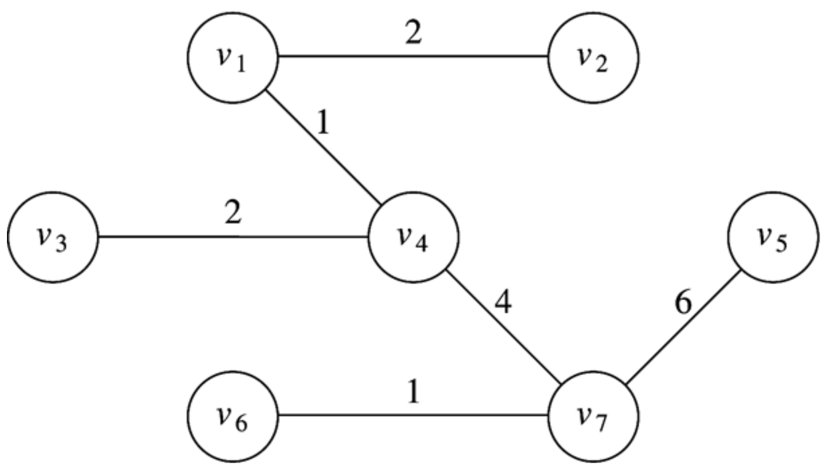
- A tree is an acyclic, undirected, connected graph
- A spanning tree of a graph is a tree containing all vertices from the graph
- A minimum spanning tree is a spanning tree, where the sum of the weights on the tree's edges are minimal

Minimum Spanning Tree

Graph:



MST:



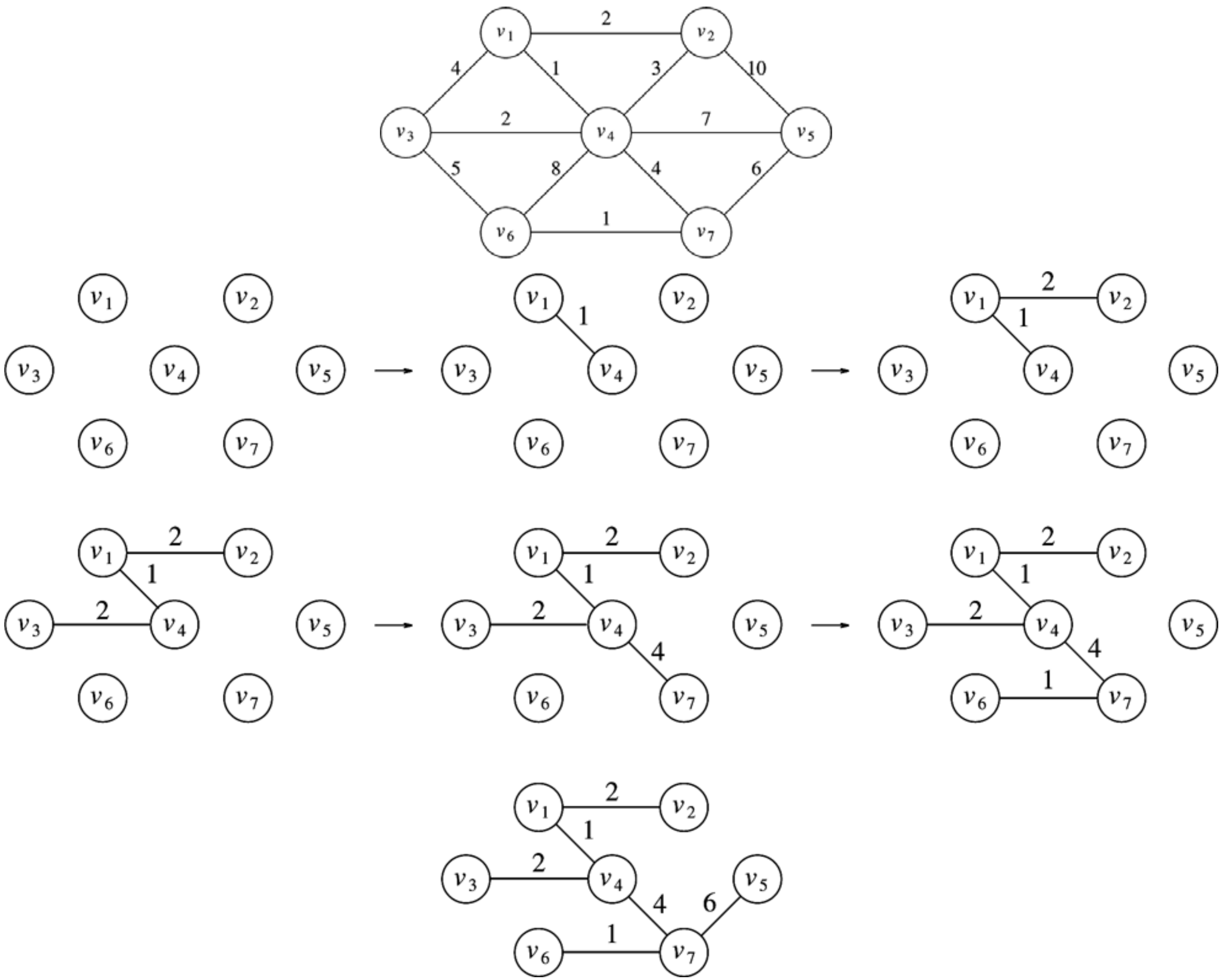
Minimum Spanning Tree

- Problem
  - Given an undirected, weighted graph $G=(V,E)$ with weights $w(u,v)$ for each $(u,v) \in E$
  - Find an acyclic, connected graph $G'=(V,E')$, $E' \subseteq E$, that minimizes $\Sigma_{(u,v) \in E'} w(u,v)$
  - G' is a minimum spanning tree
    - There can be more than one

Minimum Spanning Tree

- Solution #1
  - Start with an empty tree T
  - While T is not a spanning tree
    - Find the lowest-weight edge that connects a vertex in T to a vertex not in T
    - Add this edge to T
- T will be a minimum spanning tree
- Called Prim's algorithm (1957)

Prim’s Algorithm: Example



Prim's Algorithm

- Similar to Dijkstra's shortest-path algorithm
- Except
  - v.known = v in T
  - v.dist = weight of lowest-weight edge connecting v to a known vertex in T
  - v.path = last neighboring vertex changing (lowering) v's dist value (same as before)

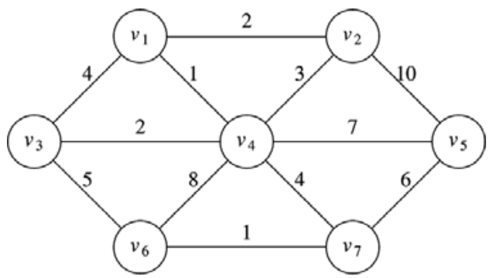
```
struct Vertex
{
    List        adj;
    bool        known;
    DistType    dist;
    Vertex      path;
        // Other data
};
```

Prim’s Algorithm

```
void Graph::dijkstra( Vertex s )
{
    for each Vertex v
    {
        v.dist = INFINITY;
        v.known = false;
    }

    s.dist = 0;

    for( ; ; )
    {
        Vertex v = smallest unknown distance vertex;
        if( v == NOT_A_VERTEX )
            break;
        v.known = true;

        for each Vertex w adjacent to v
            if( !w.known )
                if( v.dist + cvw < w.dist )
                {
                    // Update w
                    decrease( w.dist to v.dist + cvw );
                    w.path = v;
                }
    }
}
```

Running time same as Dijkstra: O(|E| log |V|) using binary heaps.

Prim's Algorithm: Example

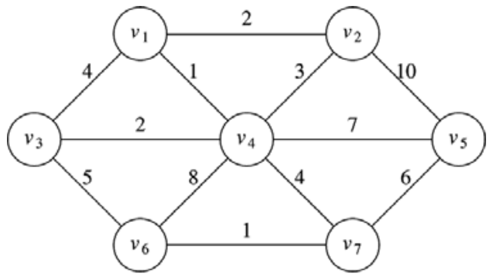


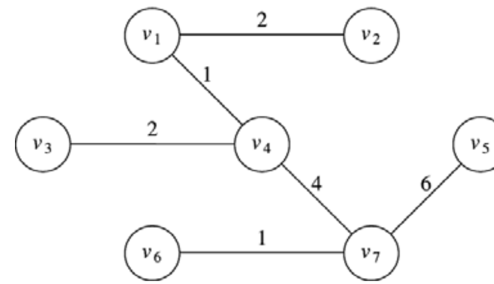
v	*known*	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	4	v_1
v_4	F	1	v_1
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	2	v_4
v_4	T	1	v_1
v_5	F	7	v_4
v_6	F	8	v_4
v_7	F	4	v_4

Prim's Algorithm: Example





v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	F	7	v_4
v_6	F	5	v_3
v_7	F	4	v_4

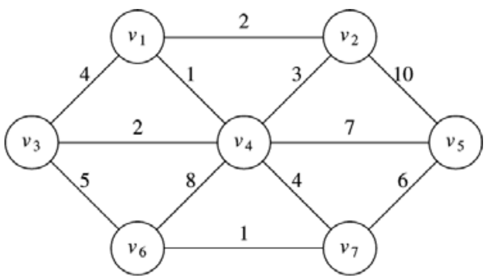
v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	F	6	v_7
v_6	F	1	v_7
v_7	T	4	v_4

v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	T	6	v_7
v_6	T	1	v_7
v_7	T	4	v_4

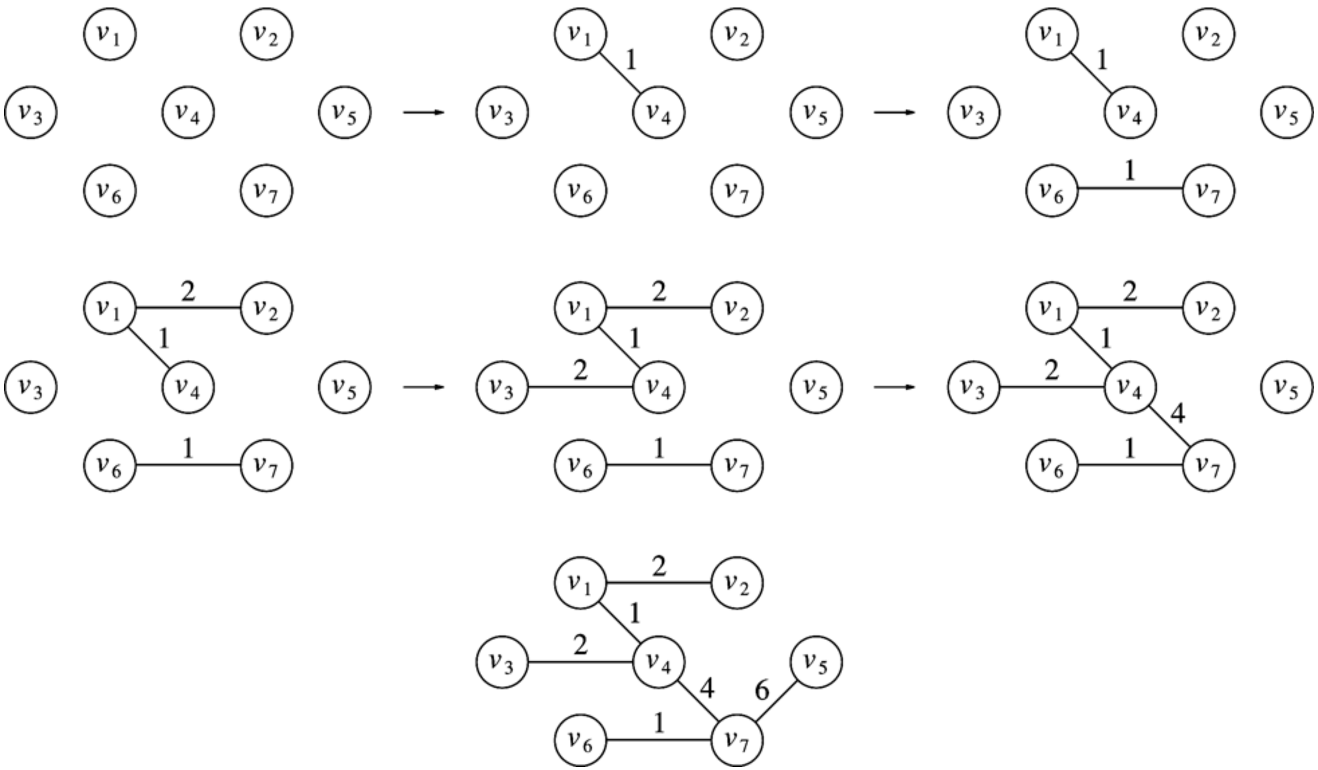
Minimum Spanning Tree

- Solution #2
  - Start with T = V (no edges)
  - For each edge in increasing order by weight
    - If adding edge to T does not create a cycle
    - Then add edge to T
- T will be a minimum spanning tree
- Called Kruskal's algorithm (1956)

Kruskal's Algorithm: Example



Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted



Kruskal’s Algorithm

```
void Graph::kruskal( )
{
    int edgesAccepted = 0;
    DisjSet ds( NUM_VERTICES );
    PriorityQueue<Edge> pq( getEdges( ) );
    Edge e;
    Vertex u, v;

    while( edgesAccepted < NUM_VERTICES - 1 )
    {
        pq.deleteMin( e );      // Edge e = (u. v)
        SetType uset = ds.find( u );
        SetType vset = ds.find( v );
        if( uset != vset )
        {
            // Accept the edge
            edgesAccepted++;
            ds.unionSets( uset, vset );
        }
    }
}
```

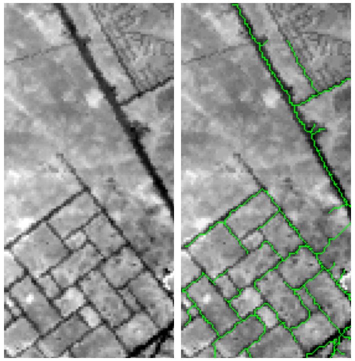
Uses Disjoint Set and Priority Queue data structures.

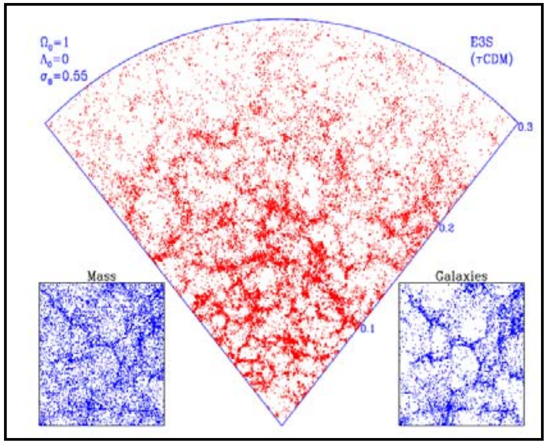
Kruskal's Algorithm: Analysis

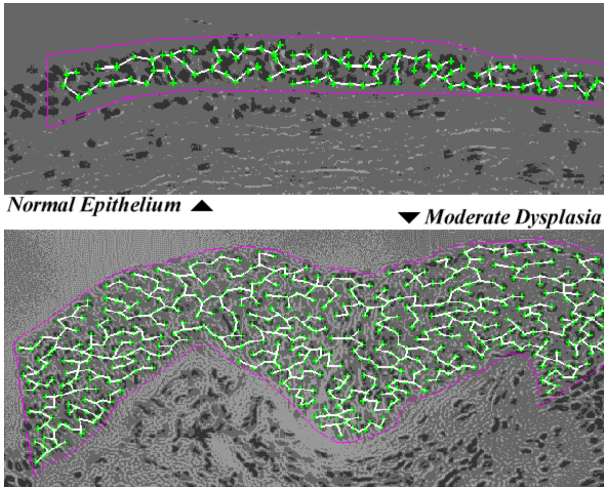
- Worst case: $O(|E| \log |E|)$
- Since $|E| = O(|V|^2)$, worst case also $O(|E| \log |V|)$
  - Running time dominated by heap operations
- Typically terminates before considering all edges, so faster in practice

Minimum Spanning Tree: Applications

- Feature extraction from remote sensing images (e.g., roads, rivers, etc.)
- Cosmological structure formation
- Cancer imaging
  - Arrangement of cells in the epithelium (tissue surrounding organs)
- Approximate solution to traveling salesman problem
- Most of above use Euclidian MST
  - I.e., weights are Euclidean distances between vertices







Minimum Spanning Trees: Summary

- Finding set of edges that minimally connect all vertices
- Fast algorithm with many important applications
- Utilizes advanced data structures to achieve fast performance