Data Structure

Lecture#25: Graphs 3 (Chapter 11)

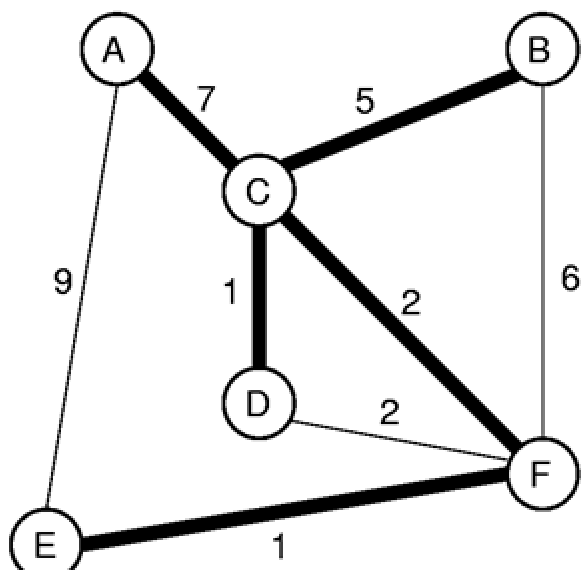
U Kang
Seoul National University

In This Lecture

- MST (Minimum Spanning Tree) problem
- Main idea and cost of Prim's algorithm for MST
- Main idea and cost of Kruskal's algorithm for MST

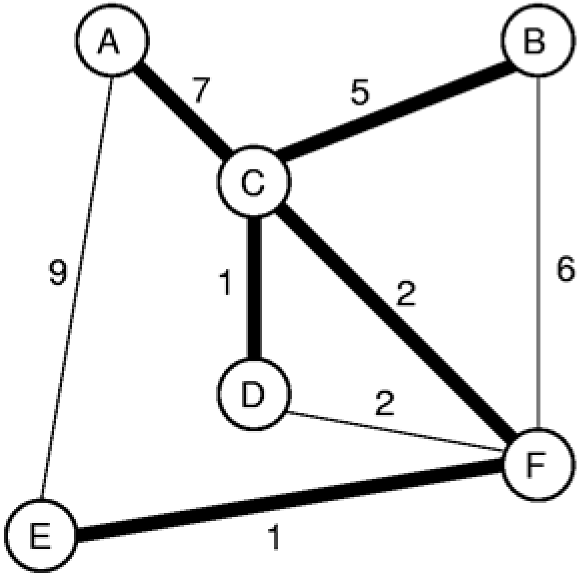
Minimum Cost Spanning Trees (1)

- Minimum Cost Spanning Tree (MST) Problem:
  - Input: An undirected, connected graph G.
  - Output: The subgraph of G that
  - 1) has minimum total cost as measured by summing the values of all the edges in the subset, and
  - 2) keeps the vertices connected.



Minimum Cost Spanning Trees (2)

- Minimum Cost Spanning Tree (MST) Problem:
  - A tree means a graph without cycle
  - Property 1) of the output ensures MST has no cycle (why?)
    - Property 1) G has minimum total cost as measured by summing the values of all the edges in the subset



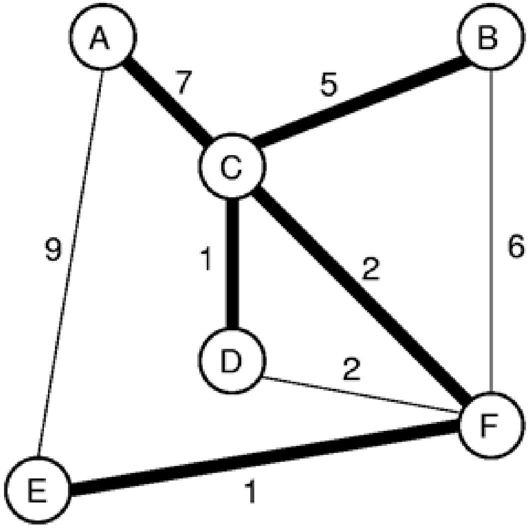
Minimum Cost Spanning Trees (3)

- Minimum Cost Spanning Tree (MST) Problem:
  - Solution may not be unique
    - E.g., (C,F) can be replaced with (D,F) in the previous slide
  - But, the minimum costs are the same for all the solutions

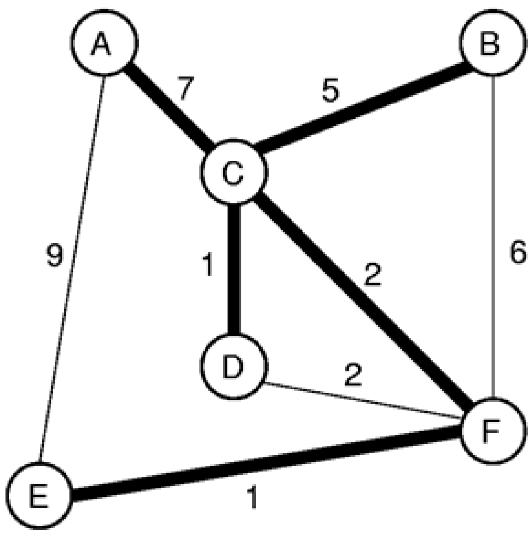
- MST algorithms
  - Prim's algorithm
  - Kruskal's algorithm

Prim's MST Algorithm (1)

- Start with any vertex N in the graph. Add N to MST.
- Pick the least-cost edge (N,M) connected to N.
- Add vertex M and edge (N,M) to the MST.
- Pick the least-cost edge connected to (N or M). Let the vertex X be connected to the edge.
  - X should not belong to the current MST.
- Add vertex X and the edge to the MST.
- (continue, until all vertices are added to MST)…



Prim's MST Algorithm (2)



	A	B	C	D	E	F
Initial	∞	∞	0	∞	∞	∞
Process C	7	5	0	1	∞	2
Process D	7	5	0	1	∞	2
Process F	7	5	0	1	1	2
Process E	7	5	0	1	1	2
Process B	7	5	0	1	1	2
Process A	7	5	0	1	1	2

Prim's MST Algorithm (3)

```
// Compute a minimal-cost spanning tree
void Prim(Graph G, int s, int[] D, int[] V) {
  int v, w;
  for (int i=0; i<G.n(); i++)   // Initialize
    D[i] = Integer.MAX_VALUE;
  D[s] = 0;
  for (int i=0; i<G.n(); i++) {
    v = minVertex(G, D);
    G.setMark(v, VISITED);
    if (v != s) AddEdgetoMST(V[v], v);
    if (D[v] == Integer.MAX_VALUE) return;
    for (w=G.first(v); w<G.n(); w=G.next(v, w))
      if (D[w] > G.weight(v, w)) {
        D[w] = G.weight(v, w);
        V[w] = v;
      }
  }
}
```

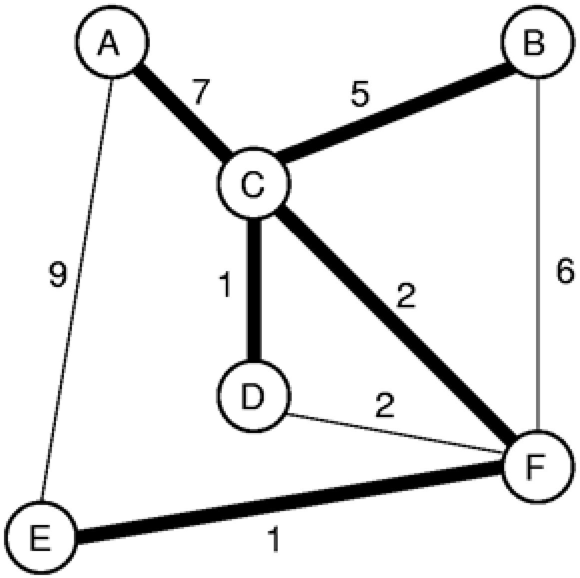
Running Time of Prim's MST

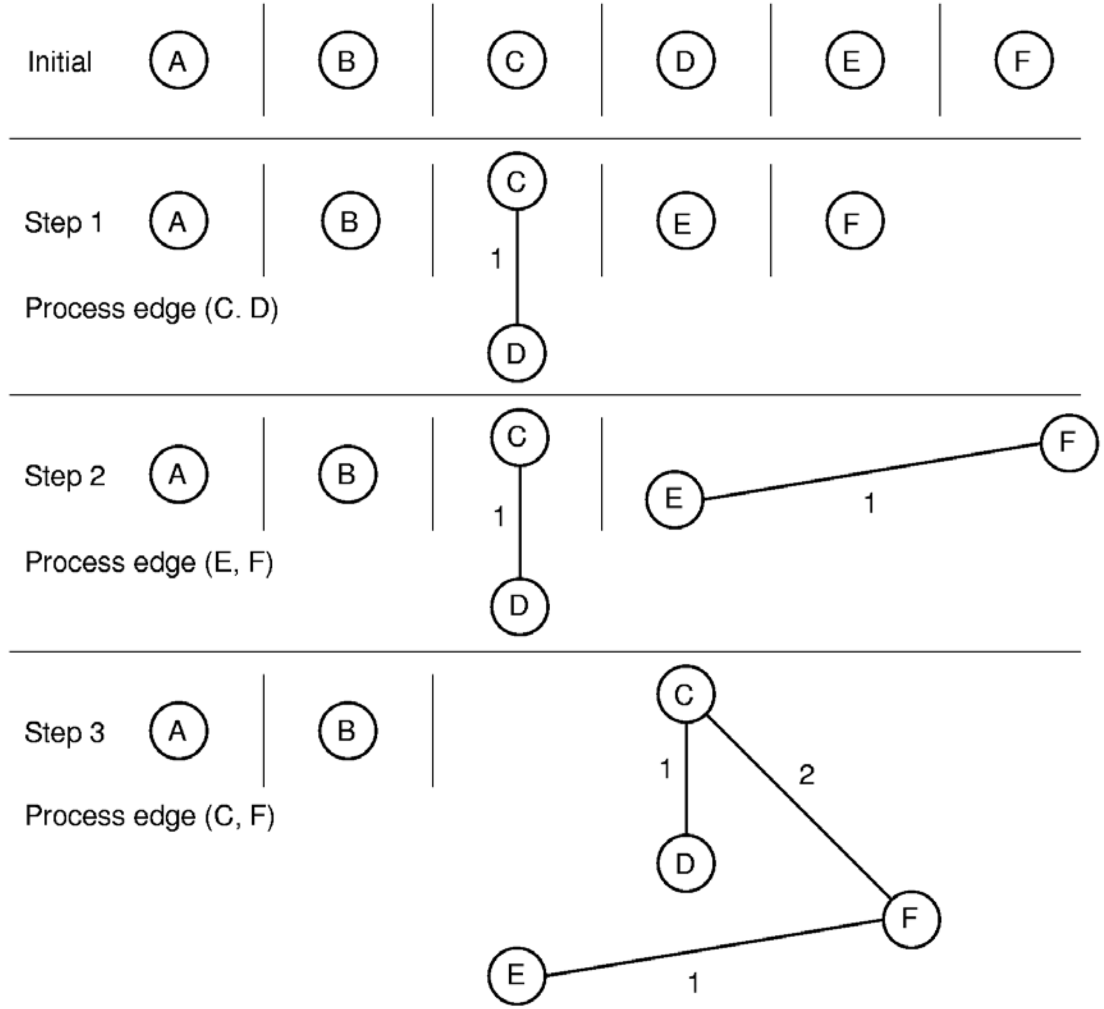
- Prim's MST is very similar to Dijkstra's algorithm
  - MST: `D[w] = G.weight(v, w);`
  - Shortest Path: `D[w] = D[v] + G.weight(v, w);`
- So does the running time of Prim's MST
  - Scan through the distance table: $\Theta(|\mathbf{V}|^2 + |\mathbf{E}|) = \Theta(|\mathbf{V}|^2)$
  - Min-heap: $\Theta((|\mathbf{V}| + |\mathbf{E}|)\log|\mathbf{V}|)$

Kruskal's MST Algorithm (1)

- Initially, each vertex is in its own MST.
- Merge two MST's that have the shortest edge between them.
  - Use a priority queue to order the unprocessed edges. Grab next one at each step.
  - Make sure the edge does not connect two vertices in a same MST
- How to tell if an edge connects two vertices already in the same MST?
  - Use the UNION/FIND algorithm with parent-pointer representation.

Kruskal's MST Algorithm (2)





Kruskal's MST Algorithm (3)

- Cost is dominated by the time to remove edges from the heap.
  - Can stop processing edges once all vertices are in the same MST

- Total cost: $\Theta(|\mathbf{E}| \log |\mathbf{E}|)$
  - Can remove edges $|E|$ times

Summary

- MST (Minimum Spanning Tree) problem
- Main idea and cost of Prim's algorithm for MST
  - Cost the same as that of Dijkstra's
- Main idea and cost of Kruskal's algorithm for MST

Questions?