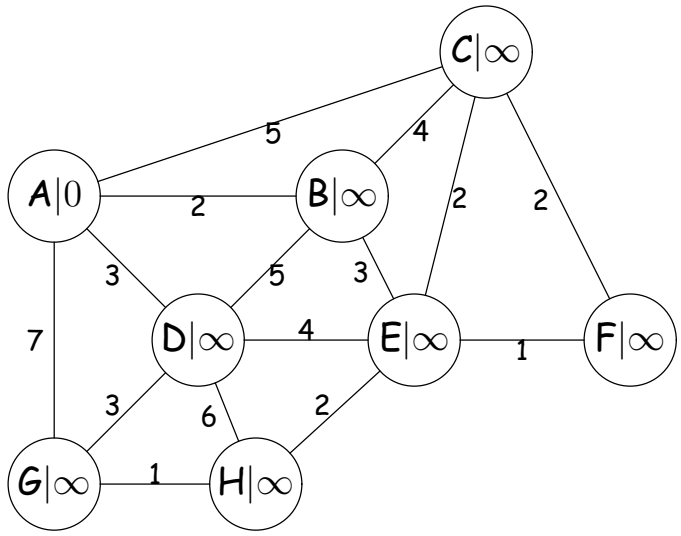
Minimum Spanning Trees by Prim's Algorithm

- Idea is to grow a tree starting from an arbitrary node.
- At each step, add the shortest edge connecting some node already in the tree to one that isn't yet.
- Why must this work?

```
PriorityQueue fringe;
For each node v { v.dist() = ∞; v.parent() = null; }
Choose an arbitrary starting node, s;
s.dist() = 0;
fringe = priority queue ordered by smallest .dist();
add all vertices to fringe;
while (! fringe.isEmpty()) {
  Vertex v = fringe.removeFirst ();

  For each edge (v,w) {
    if (w ∈ fringe && weight(v,w) < w.dist())
      { w.dist() = weight (v, w); w.parent() = v; }
  }
}
```



Minimum Spanning Trees by Kruskal's Algorithm

- Observation: the shortest edge in a graph can always be part of a minimum spanning tree.
- In fact, if we have a bunch of subtrees of a MST, then the shortest edge that connects two of them can be part of a MST, combining the two subtrees into a bigger one.
- So,...

```
Create one (trivial) subtree for each node in the graph;
MST = {};

for each edge (v,w), in increasing order of weight {
   if ( (v,w) connects two different subtrees ) {
      Add (v,w) to MST;
      Combine the two subtrees into one;
   }
}
```

Recursive Depth-First Traversal

- Previously, we saw an iterative way to do depth-first traversal of a graph from a particular node.
- We are often interested in traversing all nodes of a graph, so we can repeat the procedure as long as there are unmarked nodes.
- Recursive solution is also simple:

```
void traverse (Graph G) {
   for (v ∈ nodes of G) {
      traverse (G, v);
}

void traverse (Graph G, Node v) {
   if (v is unmarked) {
     mark (v);
     visit v;
     for (Edge (v, w) ∈ G)
       traverse (G, w);
   }
}
```

Another Take on Topological Sort

- Observation: if we do a depth-first traversal on a DAG whose edges are reversed, and execute the recursive `traverse` procedure, we finish executing `traverse(G,v)` in proper topologically sorted order.

```
void topologicalSort (Graph G) {
   for (v ∈ nodes of G) {
      traverse (G, v);
}

void traverse (Graph G, Node v) {
   if (v is unmarked) {
     mark (v);
     for (Edge (w, v) ∈ G)
       traverse (G, w);
     add v to the result list;
   }
}
```