Spanning trees

Goldsmiths Computing

Motivation

- Internet routing
- Electricity, cable, road networks
- Maze generation

Definition

A tree T is a spanning tree of a graph G if it is:

a subgraph of G includes only edges that are present in G; and

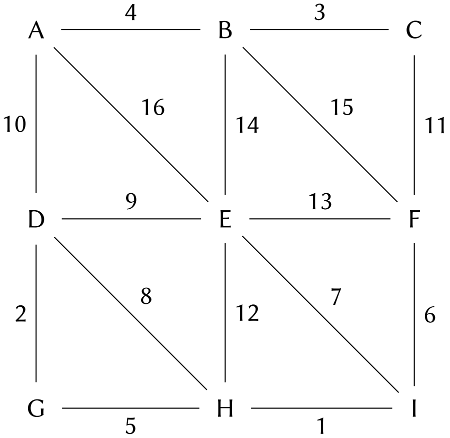
spans G includes all vertices of G

Properties

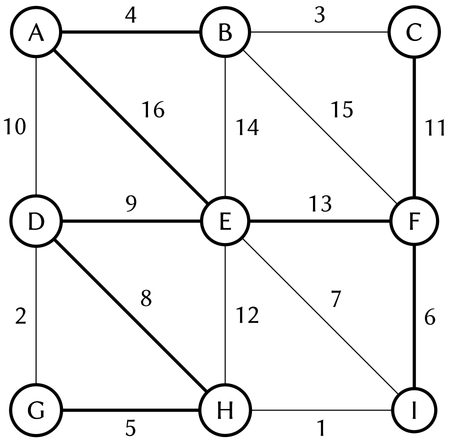
If G has $|V|$ vertices, a spanning tree has:

- $|V|$ vertices
- $|V|-1$ edges (proof?)

Random spanning tree



Random spanning tree



Minimum spanning tree

A minimum spanning tree for graph G is a spanning tree of graph G whose edge weights sum to the minimum possible total weight for that graph.

Prim's algorithm

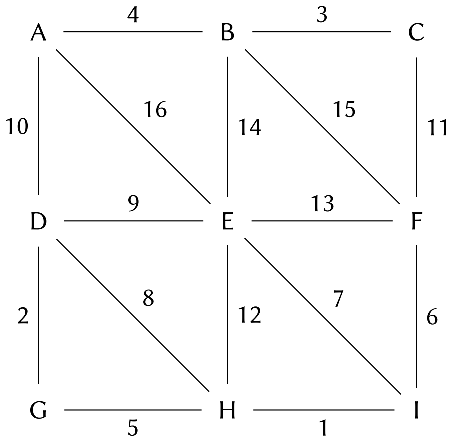
```
function PrimMST(G)
    vs ← vertices(G)
    T ← new Graph(first(vs),{})
    while |T| < |G| do
        E ← {e | e ∈ edges(G) ∧
            from(e) ∈ vertices(T) ∧ to(e) ∉ vertices(T)}
        newE ← argmin_{e∈E} weight(e)
        newV ← to(newE)
        addVertex(T,newV); addEdges(T,newE)
    end while
end function
```

1. Initialise the minimum spanning tree with a vertex from the graph.
2. Until all the vertices are included in the tree,
    - find the edge with smallest weight that links a vertex in the tree so far with a vertex not yet in the tree;
    - add that edge, and the new vertex, to the minimum spanning tree.

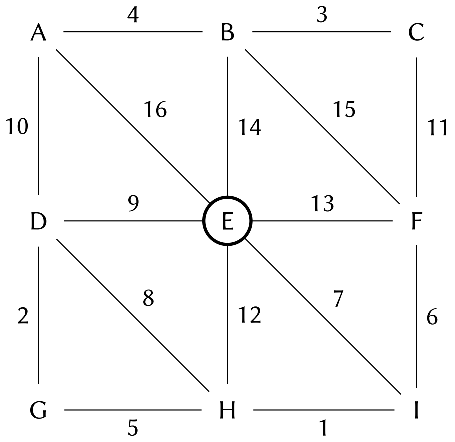
Prim's algorithm

```
function PrimMST(G)
    vs ← vertices(G)
    T ← new Graph(first(vs),{})
    while |T| < |G| do
        newE ← NIL; newV ← NIL; w ← ∞
        for e ∈ edges(G) ∧ from(e) ∈ vertices(T) ∧ to(e) ∉ vertices(T) do
            if weight(e) < w then
                w ← weight(e); newE ← e; newV ← to(e)
            end if
        end for
        addVertex(T,newV); addEdge(T,newE)
    end while
end function
```

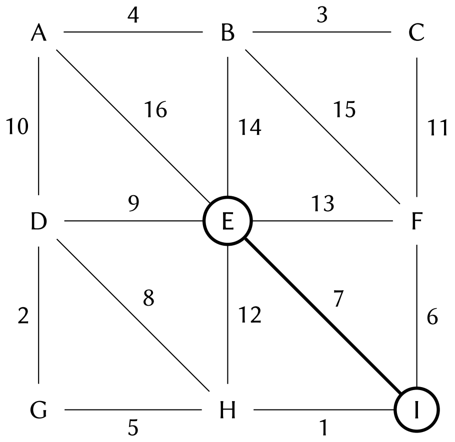
Prim’s algorithm: example



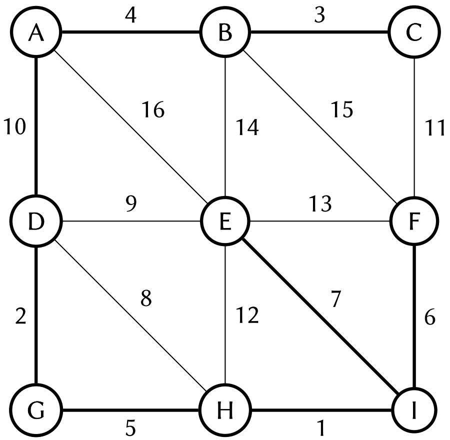
Prim's algorithm: example



Prim's algorithm: example



Prim’s algorithm: example



Prim's algorithm: proof sketch

By contradiction. Let P be the spanning tree generated by Prim's algorithm on G, and T be the minimum spanning tree.

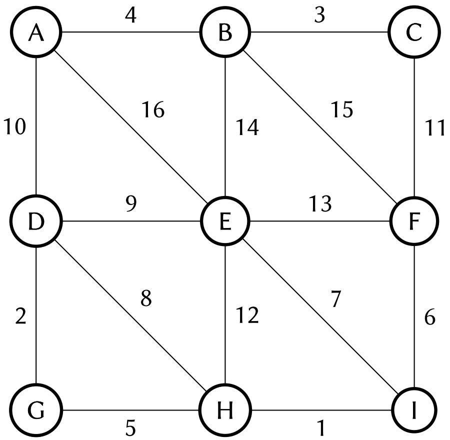
- if P = T, we are done.
- if P ≠ T,
  - there is an edge e in P not in T;
  - we added that edge to P at some point, joining a set of vertices V in Prim's tree with one of the set of vertices G-V;
  - find the edge f in T that joins V with G-V;
  - the tree T-f+e must have lower cost than T (why?) and is a spanning tree (why?).

Kruskal's algorithm

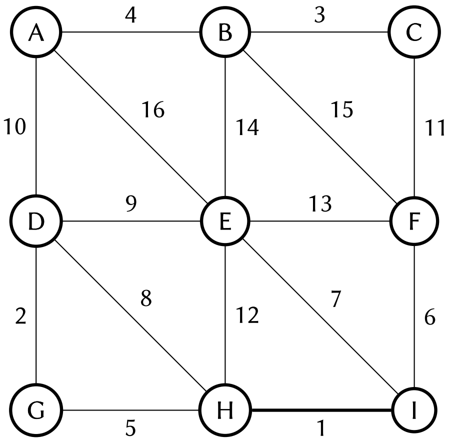
```
function KruskalMST(G)
    vs ← vertices(G)
    T ← new Graph(vs,{})
    Z ← new DisjointSet()
    for v ∈ vs do
        make-set(Z,v)
    end for
    for (u,v) ∈ edges(G) sorted by weight do
        if find(Z,u) ≠ find(Z,v) then
            addEdge(T,(u,v))
            union(Z,u,v)
        end if
    end for
end function
```

1. Initialise the minimum spanning tree with all vertices and no edges.
2. Put each vertex into its own equivalence class
3. Iterate over all the edges in the graph, ordered by weight:
    - add the edge to the tree if it joins two different equivalence classes;
    - make the edge's two vertices' equivalence classes be the same

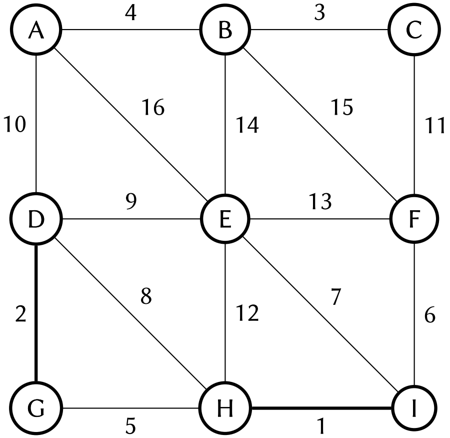
Kruskal's algorithm: example



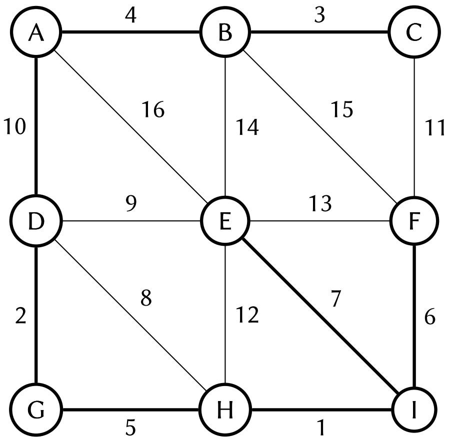
Kruskal's algorithm: example



Kruskal's algorithm: example



Kruskal’s algorithm: example



Kruskal's algorithm: proof

Similar to Prim's algorithm: consider the first edge added in the spanning tree K that is not in T

Work

1. Reading:
   - CLRS, chapter 23
   - Drozdek, section 8.5
   - DPV, section 5.1
2. Questions:

   CLRS Exercises 23.1-7, 23.2-1, 23.2-2, 23.2-4, 23.2-5

   CLRS Problem 23-1

   DPV Exercises 5.1, 5.2, 5.5
3. Complete the proof of Kruskal's algorithm.
4. Choose a concrete representation for graphs and edge sets. For your choices, what is the time complexity of Prim's algorithm?