Kruskal's Algorithm

- First step is to sort all edges by weight.
- Second step involves creation of *clusters*
  - Every vertex is initially placed in a trivial *cluster* -- the cluster for a vertex v, denoted C(v), is simply {v}. A cluster represents a local minimum spanning tree.
  - When the next edge (u,v) is considered, C(u) and C(v) are compared -- if different, (u,v) is included as an edge in the final output tree, and C(u) and C(v) are merged.

Kruskal's Algorithm

Input: A simple connected weighted graph $G = (V, E)$ with n vertices and m edges
Output: A minimum spanning tree T of G

The Algorithm:

sort E in increasing order of edge weight

for each vertex v in G, define an elementary cluster $C(v)$ (which will grow) by $C(v) = \{v\}$

$T \leftarrow$ an empty tree // T will eventually become the minimum spanning tree

while T has fewer than $n - 1$ edges **do**

  $(u,v) \leftarrow$ next edge

  $C(v) \leftarrow$ cluster containing v

  $C(u) \leftarrow$ cluster containing u

  if $C(v) \neq C(u)$ **then**
    add edge (u,v) to T
    merge $C(u)$ and $C(v)$ (and update other clusters as needed)

return T

Correctness

We need to verify the following Facts:

1. During execution, distinct clusters are always disjoint, and for each cluster C, if T[C] is the subgraph of G whose set of vertices is C and whose edges are those edges of T whose endpoints lie in C, then T[C] is a tree.
2. No cycle ever arises in T during execution of the algorithm
3. The main loop terminates (it is conceivable that after all edges have been examined, T still contains $< n - 1$ edges – this is shown to be impossible)
4. The set T that is returned is a spanning tree for G
5. The set T is a *minimum* spanning tree for G.

Verification of the Loop Invariant

To establish Facts 1 - 2, we establish the following loop invariant I(i):

(a) Distinct clusters are disjoint

(b) For each cluster C, the subgraph T[C] is a tree

(c) T contains no cycles

[NOTE: As the algorithm proceeds, we do *not* expect T itself to be a tree because it is composed of disconnected pieces. We will show that these pieces do assemble into a tree by the time the algorithm has completed.]

At the beginning of execution, when singleton clusters are first formed, (a) – (c) hold. We now assume (a) – (c) hold and consider the iteration in which the next edge (x,y) is examined, and show that (a) – (c) continue to hold. In the algorithm, if C(x) = C(y), the iteration ends and the state of the clusters and that of the set T are unchanged, so we assume $C(x) \neq C(y)$.

To establish that (a) – (c) hold, we make use of background facts from the lab.

Verifying the Loop Invariant (a)-(c)

For (a) [distinct clusters are disjoint], joining two clusters with an edge, forming a new larger cluster, does not change the fact that distinct clusters are disjoint.

For (b) [T[C] is a tree], adding the edge (x,y) to the union of the trees $T[C_x]$, $T[C_y]$ results in another tree, $T[C_x] \cup T[C_y] \cup \{(x,y)\}$, by Background Fact B (previous slide). So, if we let $C = C_x \cup C_y$, then T[C] is a tree.

For (c) [no cycles], notice that the edges that compose T itself are formed as the union of the edges in the (disjoint) trees $T[C_v]$, for v in V. Since none of these trees contains a cycle, T itself does not contain a cycle either.

This establishes the loop invariant and therefore Facts 1, 2.

Correctness – Fact 3, "while loop terminates"

We verify that the while loop in the algorithm eventually terminates:

Assume that after all edges have been examined, T still has < n-1 edges (this would be the situation when the while loop fails to terminate).

By Fact 2, T contains no cycle. By Background Fact 3, since |T| < n – 1 and contains no cycle, there is an edge e in G so that T U {e} also contains no cycles and so that e $\notin$ T.

But this situation is impossible: During the execution of Kruskal's algorithm, the edge e = (x,y) was examined at some point, because, if the while loop does not terminate, *every* edge would be examined eventually. When e was examined, Kruskal rejected e (since e was not in T after all edges had been examined); the only reason Kruskal has for rejecting is to avoid creation of a cycle in T. But, by the way e was chosen above, clearly e does not introduce a cycle, as Background Fact 3 shows.

The reasoning shows that the assumption that the while loop never terminates is incorrect.

Correctness – Fact 4, "T is a spanning tree"

We verify Fact 4, that, after execution of Kruskal's algorithm, T is a spanning tree.

- We have already seen that T has no cycles and therefore (by the way the while loop is defined) T has n – 1 edges, where n is the number of vertices in G. By the lemma below, T must have n vertices. It follows that T is a spanning tree.

Lemma. If a graph H is a graph that has n vertices and m edges and if $m \geq n$, then H has a cycle.

Proof. If H is connected but has no cycle, it follows that $m = n - 1$. Since $m \geq n$, it therefore follows that H has a cycle.

If H is not connected, write H as a union of its connected components: $H = H_1 \cup H_2 \cup \ldots \cup H_k$, where $k > 1$, where, for each i, H_i has n_i vertices and m_i edges. Since each H_i is a tree, $m_i = n_i - 1$. We have therefore

$$m = m_1 + \ldots + m_k = (n_1 - 1) + \ldots + (n_k - 1) = n - k < n,$$

contradiction.

Correctness – Fact 5

To verify Fact 5 – that, after execution of Kruskal's algorithm, T is a *minimum* spanning tree – we establish the following loop invariant I(i):

At each stage i of the algorithm, if T is the collection of edges obtained so far, there is a minimum spanning tree for G that contains W.

Assuming we can establish this loop invariant, then, when the algorithm finishes, it will follow from the loop invariant that there is an MST that contains T. But, as shown in previous slides, when the while loop of the algorithm ends, T has become a spanning tree. It follows that the MST that contains T must be T itself.

Note: Our intuition that the edges built up in any cluster form an MST for the subgraph induced by that cluster is correct, but it does not lead to a straightforward proof of Fact 5. However, having established Fact 5, we can prove the intuition is correct: Let G[C] be the induced subgraph at any stage of the algorithm and let T[C] be the edges generated by the algorithm. If we follow the algorithm, restricted to just G[C], we will conclude (as we do in the proof of Fact 5) that T[C] is an MST for G[C].

Proof of Loop Invariant

The invariant holds at the start – G must have an MST since, as we have shown, it does have a spanning tree, so one such spanning tree must have minimal weight.

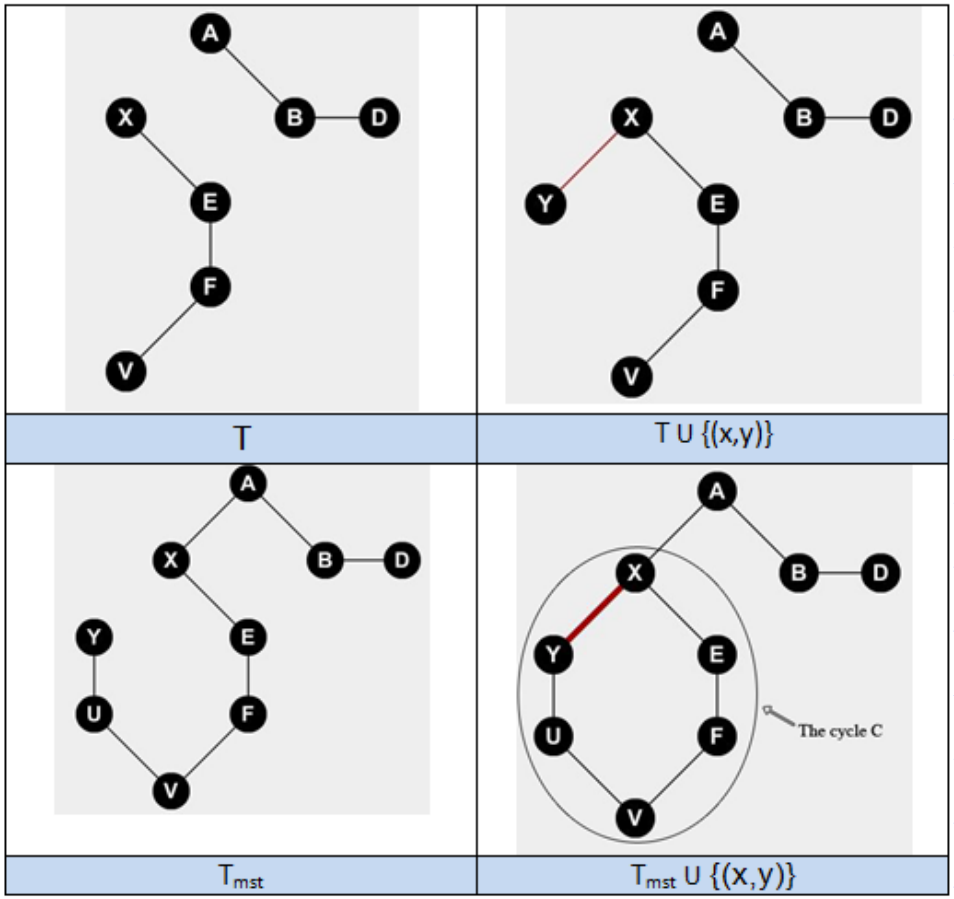
Assuming the invariant holds so far, we consider the next step of the algorithm in which the edge (x,y) is considered; WLOG, assume that $C(x) \neq C(y)$, so that the algorithm will add (x,y) to T, the set of edges obtained so far. By induction hypothesis, there is an MST for G that contains T; let us denote this MST T_{mst}. We show there is an MST for G that contains $T \cup \{(x,y)\}$.

Proof (continued)

Case I. The new edge (x,y) happens to belong to T_{mst}. In that case, T_{mst} also contains T U {(x,y)}, and the induction step is complete.

Case II. The new edge (x,y) does not belong to T_{mst}. Since T_{mst} is a spanning tree, both x and y are vertices in T_{mst}. Therefore T_{mst} U {(x,y)} contains n edges, and so contains a cycle C, with (x,y) as one of its edges (if (x,y) were not one of the edges of C, then C would be a subgraph of T_{mst}). Recall T U {(x,y)} contains no cycle (by construction). So there must be some (u,v) in C (and also in T_{mst}) that does not belong to T (since edges of C are not a subset of the edges of T). Define T' by

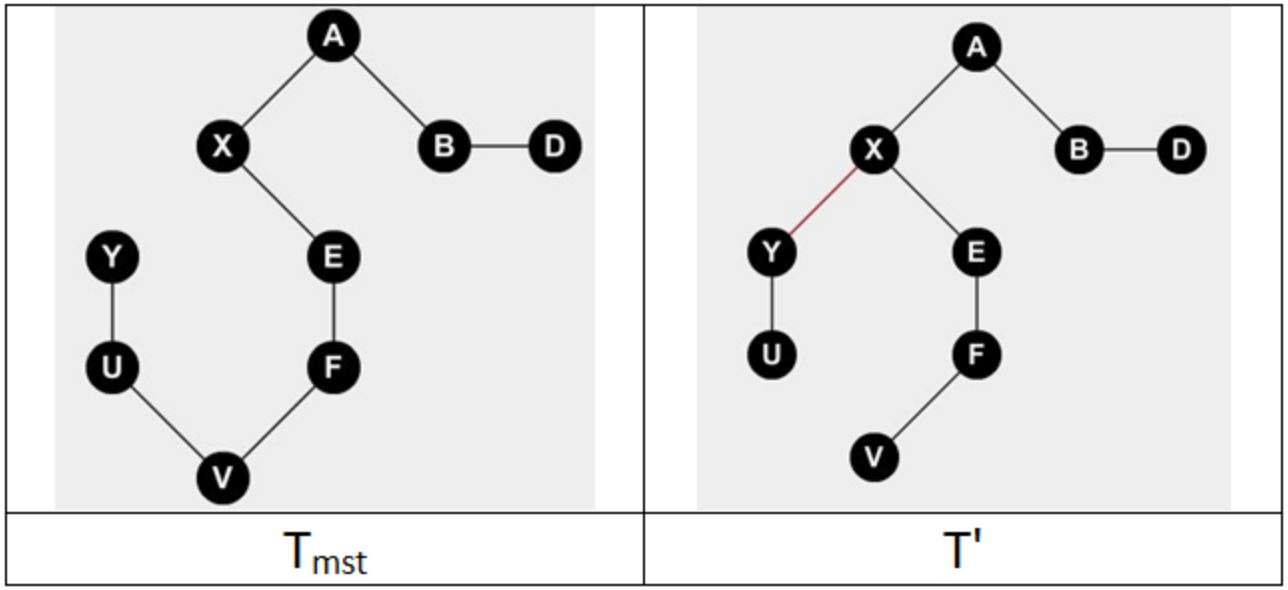
$$T' = T_{mst} \cup \{(x,y)\} - \{(u,v)\}$$



Proof (continued)

Claim 1. T' is a spanning tree.

Proof. Suppose r, s are vertices in G. Since T is a spanning tree, there is a path p from r to s. If this path begins like this: r, . . ., v, u, . . ., then we can replace the occurrence of v,u with a path in C from v back to u, and then continue to follow p. Likewise if it begins r, . . ., u, v, … Therefore, T' is connected and clearly has n – 1 edges (since T has that many); it follows that T' is a tree that visits every vertex.



Proof (continued)

Claim 2. T' = T_{mst} U {(x,y)} – {(u,v)} has the same weight as T_{mst}.

Proof. Note that T U {(u,v)} contains no cycle, since T_{mst} includes T U{(u,v)}. So the algorithm could not have already rejected (u,v) [it would reject only if (u,v) would introduce a cycle into T]. Since edges are ordered by weight, this means that (u,v) must have weight at least greater than or equal to wt(x,y). It follows that wt(T') □ wt(T_{mst}). But T_{mst} is an MST, so we also have wt(T_{mst}) □ wt(T'). This proves
Claim 2.

Continuation of Proof of Fact 5. Claims 1 and 2 show that T' is also an MST, but now T' includes both T and (x,y). This establishes the induction step of the proof of the loop invariant I. As observed earlier, we may now conclude that Fact 5 holds and that the algorithm produces an MST.