Addendum to D2: A proof that Kruskal's Algorithm works

Text Reference (Epp) 5ed: Section 10.6

Some familiar definitions

A **tree** is a connected graph with no circuits other that the trivial ones.

A **forest** is a disjoint collection of trees.

Note that we may consider a tree to be a forest (with just one tree). We shall do this. So a forest is a graph with no circuits.

Recall: Characterising Trees (D2 Slide 91)

Theorem: Let T be a graph with n vertices. The following statements are logically equivalent:

(i) T is a tree (it is connected and has no non-trivial circuits).

(ii) T has no simple circuits and $n-1$ edges.

(iii) T is connected and has $n-1$ edges.

(iv) T is connected and every edge is a bridge.

(v) Any two vertices of T are connected by exactly one simple path.

(vi) T contains no non-trivial circuits, but the addition of any new edge (connecting an existing pair of vertices) creates a simple circuit.

Kruskal's algorithm for minimal spanning tree

Kruskal's algorithm is an obvious modification to the spanning-tree-finding algorithm we discussed previously. We simply choose the 'cheapest' possible edge at each step:

Kruskal's algorithm for minimal spanning tree

Kruskal's algorithm is an obvious modification to the spanning-tree-finding algorithm we discussed previously. We simply choose the 'cheapest' possible edge at each step:

Input: Weighted connected graph G with n vertices.

Output: Minimal spanning tree T for G.
Total weight W of this tree.

Kruskal's algorithm for minimal spanning tree

Kruskal's algorithm is an obvious modification to the spanning-tree-finding algorithm we discussed previously. We simply choose the 'cheapest' possible edge at each step:

Input: Weighted connected graph G with n vertices.

Output: Minimal spanning tree T for G.
Total weight W of this tree.

Method:

1. Initialise T to have all the vertices of G but no edges. Initialise W to 0.

Kruskal's algorithm for minimal spanning tree

Kruskal's algorithm is an obvious modification to the spanning-tree-finding algorithm we discussed previously. We simply choose the 'cheapest' possible edge at each step:

Input: Weighted connected graph G with n vertices.

Output: Minimal spanning tree T for G.
Total weight W of this tree.

Method:

1. Initialise T to have all the vertices of G but no edges. Initialise W to 0.
2. From the edges currently in G pick one, e, of least weight and remove it from G.

Kruskal's algorithm for minimal spanning tree

Kruskal's algorithm is an obvious modification to the spanning-tree-finding algorithm we discussed previously. We simply choose the 'cheapest' possible edge at each step:

Input: Weighted connected graph G with n vertices.

Output: Minimal spanning tree T for G.
Total weight W of this tree.

Method:

1. Initialise T to have all the vertices of G but no edges. Initialise W to 0.
2. From the edges currently in G pick one, e, of least weight and remove it from G.
3. If adding e to T does not create a circuit in T, add e to T and add weight(e) to W.

Kruskal's algorithm for minimal spanning tree

Kruskal's algorithm is an obvious modification to the spanning-tree-finding algorithm we discussed previously. We simply choose the 'cheapest' possible edge at each step:

Input: Weighted connected graph G with n vertices.

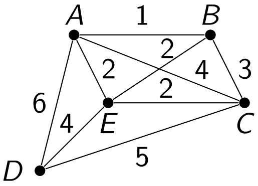
Output: Minimal spanning tree T for G.
Total weight W of this tree.

Method:

1. Initialise T to have all the vertices of G but no edges. Initialise W to 0.
2. From the edges currently in G pick one, e, of least weight and remove it from G.
3. If adding e to T does not create a circuit in T, add e to T and add weight(e) to W.
4. Repeat steps 2 and 3 until T has $n - 1$ edges.

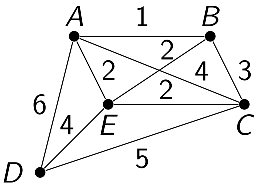
Example: Applying Kruskal's algorithm

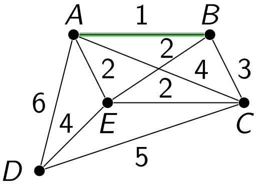
Find a minimal spanning tree for this weighted graph:



Example: Applying Kruskal's algorithm

Find a minimal spanning tree for this weighted graph:

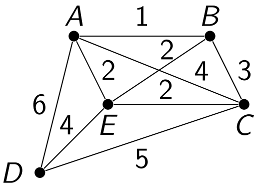


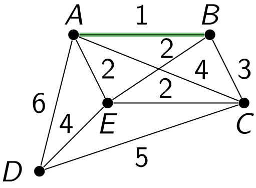


$W = 1$

Example: Applying Kruskal's algorithm

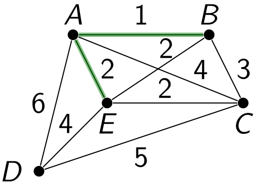
Find a minimal spanning tree for this weighted graph:





$W = 1$

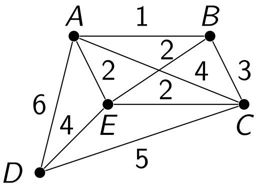
$\Longrightarrow$

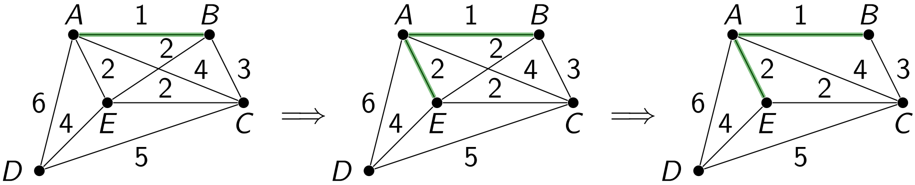


$W = 3$

Example: Applying Kruskal's algorithm

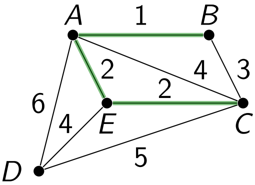
Find a minimal spanning tree for this weighted graph:





$W = 1$ $\quad\Longrightarrow\quad$ $W = 3$ $\quad\Longrightarrow\quad$ $W = 3$

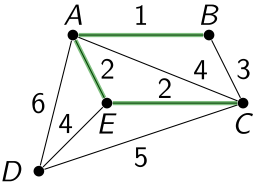
Example: Applying Kruskal's algorithm (cont.)

$\Longrightarrow$ 

$W = 5$

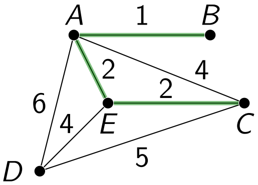
Example: Applying Kruskal's algorithm (cont.)

$\Longrightarrow$



$W = 5$

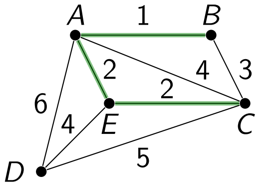
$\Longrightarrow$



$W = 5$

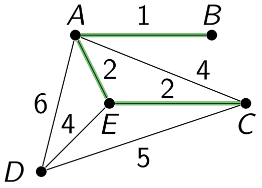
Example: Applying Kruskal's algorithm (cont.)

$\Longrightarrow$



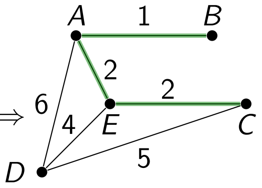
$W = 5$

$\Longrightarrow$



$W = 5$

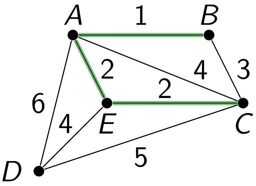
$\Longrightarrow$



$W = 5$

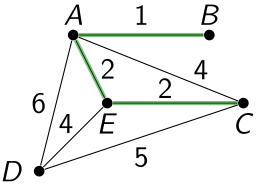
Example: Applying Kruskal's algorithm (cont.)

$\Longrightarrow$



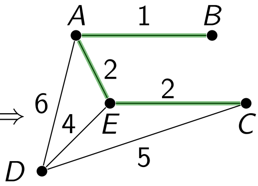
$W = 5$

$\Longrightarrow$



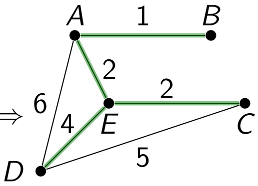
$W = 5$

$\Longrightarrow$



$W = 5$

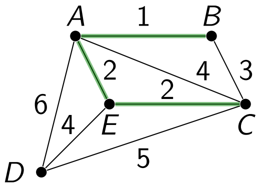
$\Longrightarrow$



$W = 9$

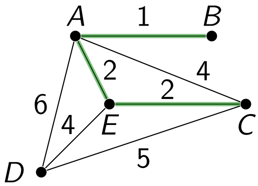
Example: Applying Kruskal's algorithm (cont.)

$\Longrightarrow$



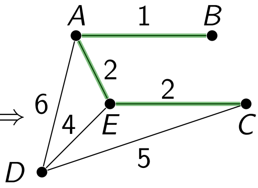
$W = 5$

$\Longrightarrow$



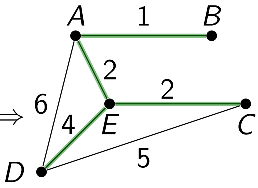
$W = 5$

$\Longrightarrow$



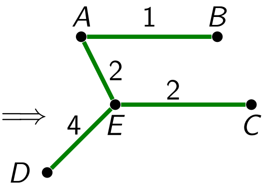
$W = 5$

$\Longrightarrow$



$W = 9$

$\Longrightarrow$



$W = 9$

(The generated spanning tree)

Modified Kruskal's algorithm for minimal spanning tree

Input: Weighted connected graph G with n vertices.

Output: Minimal spanning tree T for G.
Total weight W of this tree.

Method:

1. Initialise T to have all the vertices of G but no edges. Initialise W to 0.
2. From the edges currently in G pick one, e, of least weight and remove it from G.
3. If adding e to T does not create a circuit in T, add e to T and add weight(e) to W.
4. Repeat steps 2 and 3 **until G has no edges left to consider**.

A lemma and a theorem

Lemma: Suppose that G is a connected, weighted graph. The output from Kruskal's algorithm on input G is the same as the output of the modified Kruskal's algorithm on input G.

Theorem: (The correctness of Kruskal's Algorithm) Suppose that G is a connected, weighted graph. The output of Kruskal's algorithm on input G is is a minimal spanning tree T of G.

Correctness of (modified) Kruskal's Algorithm

Proof: Suppose that G is a connected, weighted graph with n vertices and that $T = T_m$ is the subgraph of G produced when G is input to the modified Kruskal's algorithm. We shall prove the following, in sequence:

1. T_m is a forest
2. T_m is connected
3. T_m is a spanning tree of G
4. T_m is a minimal spanning tree of G.

By the Lemma on the previous slide, this proves the theorem.

Correctness of (modified) Kruskal's Algorithm

We introduce some notation to help us talk about the algorithm, and what things look like during the execution.

Let m be the number of edges in G. Let $(e_1, e_2, \ldots, e_m)$ be the edges of G, in sequence, such that

$$\forall i \in \{1, 2, \ldots, m-1\} \ \text{weight}(e_i) \leq \text{weight}(e_{i+1}).$$

Without loss of generality, we may assume that the modified Kruskal's algorithm considers the edges in order $e_1, e_2, \ldots$.

Let T_0 be T as it looks after step 1 (T_0 has all the vertices of G, but no edges). For $i = 1, 2, \ldots, m$, let T_i be T as it looks after step 3 has been executed i times (that is, after e_i is considered).

Correctness of the (modified) Kruskal's Algorithm

Let $p(i)$: T_i is a forest.

We use mathematical induction to prove $\forall i \in \{0, 1, \ldots, m\}\ p(i)$.

It is clear that T_0 is a graph with n vertices and 0 edges. Since T_0 has no edges, it is a forest (with n connected components). Hence $p(0)$ holds.

Let $i \in \{0, 1, \ldots, m-1\}$, and suppose that $p(0), p(1), \ldots, p(i)$ all hold. Since $p(i)$ is true, T_i is a forest. Since $i \leq m-1 < m$, step 2 of the algorithm will be executed for an $(i+1)$-th time. The edge removed from G in step 2 is e_{i+1}. Step 3 of the algorithm will be executed an $(i+1)$-th time. We consider cases, based on whether adding e_{i+1} to T_i creates a circuit.

Correctness of (modified) Kruskal's Algorithm

Consider first the case that adding e_{i+1} to T_i makes a circuit. Then $T_{i+1} = T_i$. Since T_i is a forest, T_{i+1} is a forest.

Consider next the case that adding e_{i+1} to T_i does not make a circuit. Then T_{i+1} is constructed from T_i by adding e_{i+1}. Since T_i is circuit-free, and adding e_{i+1} does not add a circuit, T_{i+1} is circuit-free. That is, T_{i+1} is a forest.

In each case, T_{i+1} is a forest. Hence $p(i+1)$ is true.

By the principle of mathematical induction, $\forall i \in \{0, 1, \ldots, m\}\ p(i)$. Since $p(m)$ is true, T_m is a forest.

Correctness of (modified) Kruskal's Algorithm

Next we show that T_m is a tree. Since we know that T_m is a forest, we need only show that T_m is connected. We shall use a proof by contradiction. Suppose that T_m is not connected. Let $F_1, F_2, \ldots, F_s$ be the connected components of G. Let $v_1 \in V(F_1)$ and let $v_2 \in V(F_2)$. Since G is connected, there exists a path α in G from v_1 to v_2. Since v_2 is not in $V(F_1)$, α visits vertices not in $V(F_1)$. Let v_3 be the first vertex visited by α that is not in $V(F_1)$. Let e_j be the edge traversed by α to reach v_3 for the first time. Then one endpoint of e_j is in F_1, and the other endpoint is not. Since the end-points of e_j lie in distinct connected components of T_m, and T_m is a forest, adding e_j to T_m will not create a circuit. Since T_{j-1} is a subgraph of T_m, adding e_j to T_{j-1} does not create a circuit. It follows that e_j will added to T_{j-1} when executing step 3 for the j-th time. But then $e_j \in T_m$. This is a contradiction. Since our supposition that T_m is not connected allowed us to deduce a contradiction, it must be false. Hence T_m is connected.

Correctness of (modified) Kruskal's Algorithm

Since T_m is a connected forest it is a tree. Since T_0 contains every vertex from G, T_m does too. Hence T_m is a spanning tree of G.

Correctness of (modified) Kruskal's Algorithm

Now we show that T_m is a mimimal spanning tree. We shall use a proof by contradiction. Suppose that T is not a minimal spanning tree. Let S be a minimal spanning tree for G such that the number of edges that T_m and G have in common is maximal among all minimal spanning trees (no minimal spanning tree of G shares more edges with T_m than S does—some might share different edges, but none share more). Since S is a minimal spanning tree for G and T_m is not, the set $C = \{e \in E(T_m) \mid e \notin E(S)\}$ is not empty. Let k be the minimum integer such that $e_k \in C$. Since $e_k \notin E(S)$, adding e_k to S would add a circuit. Let e_ℓ be one of the edges in that circuit such that $e_\ell \notin T_m$ (such an edge must exist because T_m is circuit-free). Let S' be the graph obtained from S by adding e_k (creating a connected graph with a circuit that contains e_ℓ) and then removing e_ℓ. Removing an edge from a circuit in a connected graph leaves a connected graph, hence S' is connected. Since S' is a connected graph with n vertices and $n-1$ edges, it is a tree.

Correctness of (modified) Kruskal's Algorithm

Suppose that weight(e_ℓ) $<$ weight(e_k). Then $\ell < k$, and e_ℓ would have been considered for inclusion in T before e_k. By our choice of k, all of the edges in T_{k-1} are in S; hence all of the edges in $T_{\ell-1}$ are in S. Since $e_\ell \in E(S)$ and S is circuit-free and $T_{\ell-1}$ is a subgraph of S, adding e_ℓ to $T_{\ell-1}$ does not create a circuit. Hence e_ℓ will be added to T when step 3 is exceuted for the ℓ-th time. Hence $e_\ell \in T_m$. But we chose $e_\ell \notin T_m$. This contradiction shows that weight(e_ℓ) $\geq$ weight(e_k).

Since S' is constructed from S by removing e_ℓ and adding e_k, and weight(e_ℓ) $\geq$ weight(e_k), we have that weight(S) $\geq$ weight(S'). Since S is a minimal spanning tree, weight(S') $=$ weight(S). It follows that S' is a minimal spanning tree of G. But S' shares one more edge with T than S does, contradicting our choice of S. Since our supposition that T is not a minimal spanning tree allowed us to deduce a contradition, it must be false. $\square$

Discussion

What was the point of that?