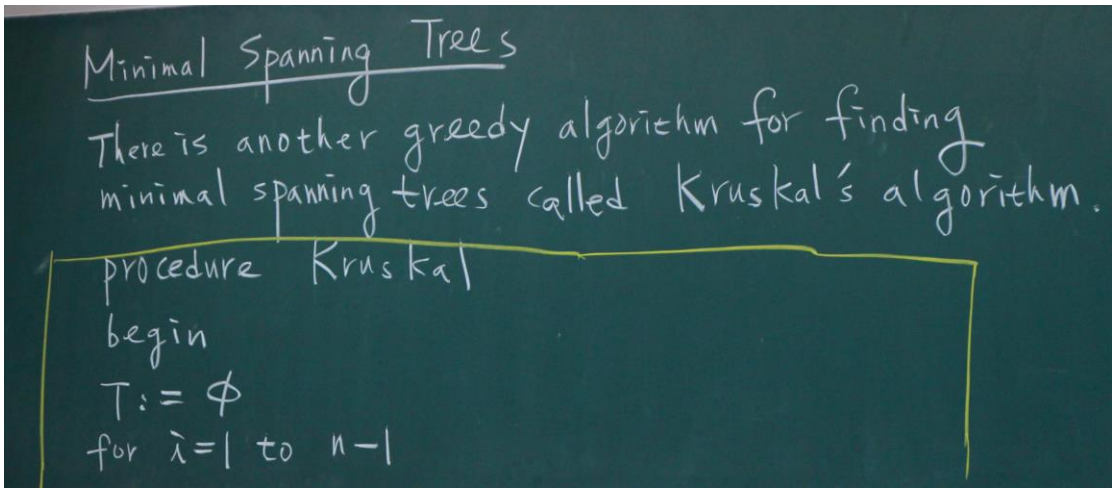
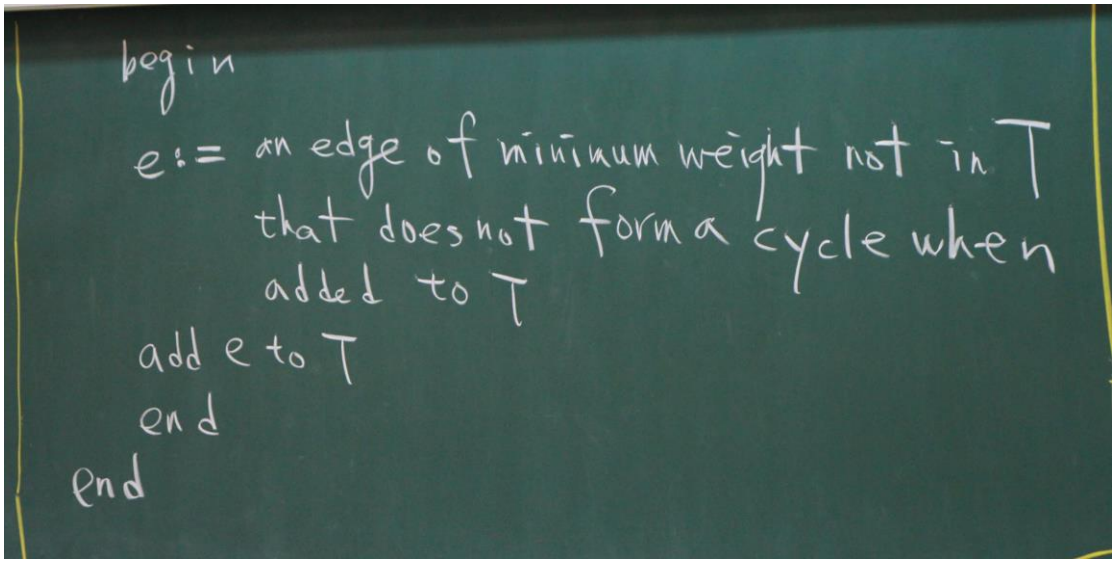
【10920 趙啟超教授離散數學 / 第 28 堂版書】





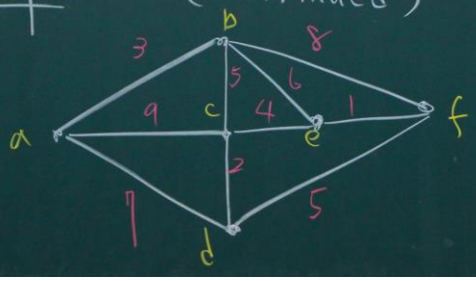
Minimal Spanning Trees

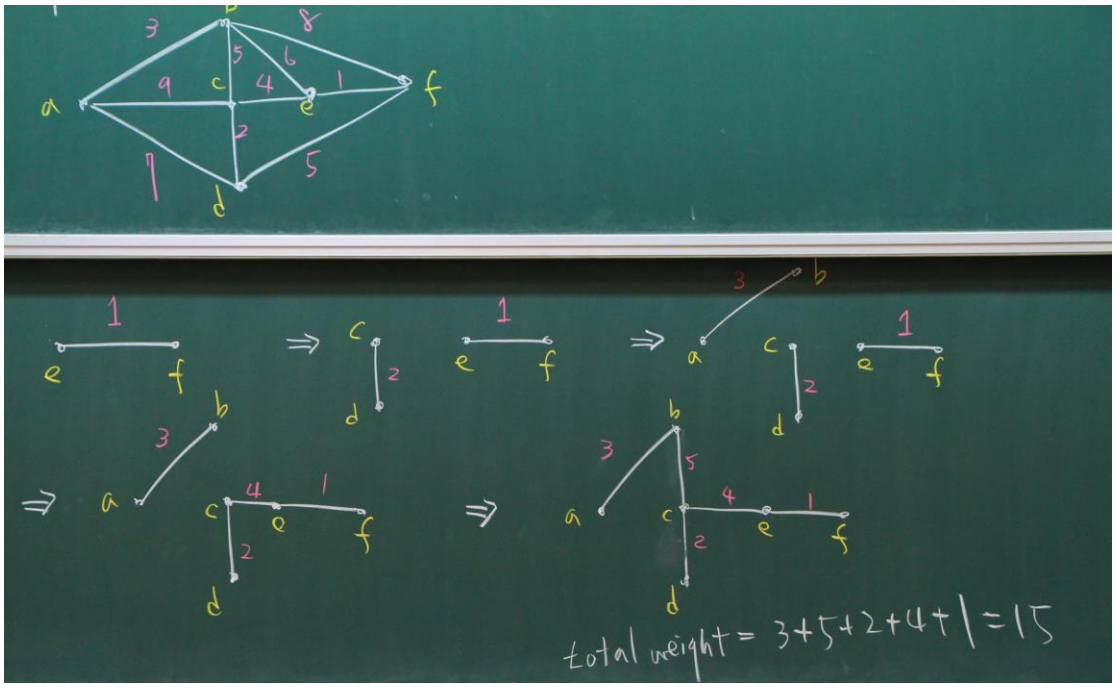
There is another greedy algorithm for finding minimal spanning trees called Kruskal's algorithm.

```
procedure Kruskal
begin
T := ∅
for i=1 to n-1
    begin
    e := an edge of minimum weight not in T
         that does not form a cycle when
         added to T
    add e to T
    end
end
```

We need to sort the edges by weight first and then examine them from smallest to largest.

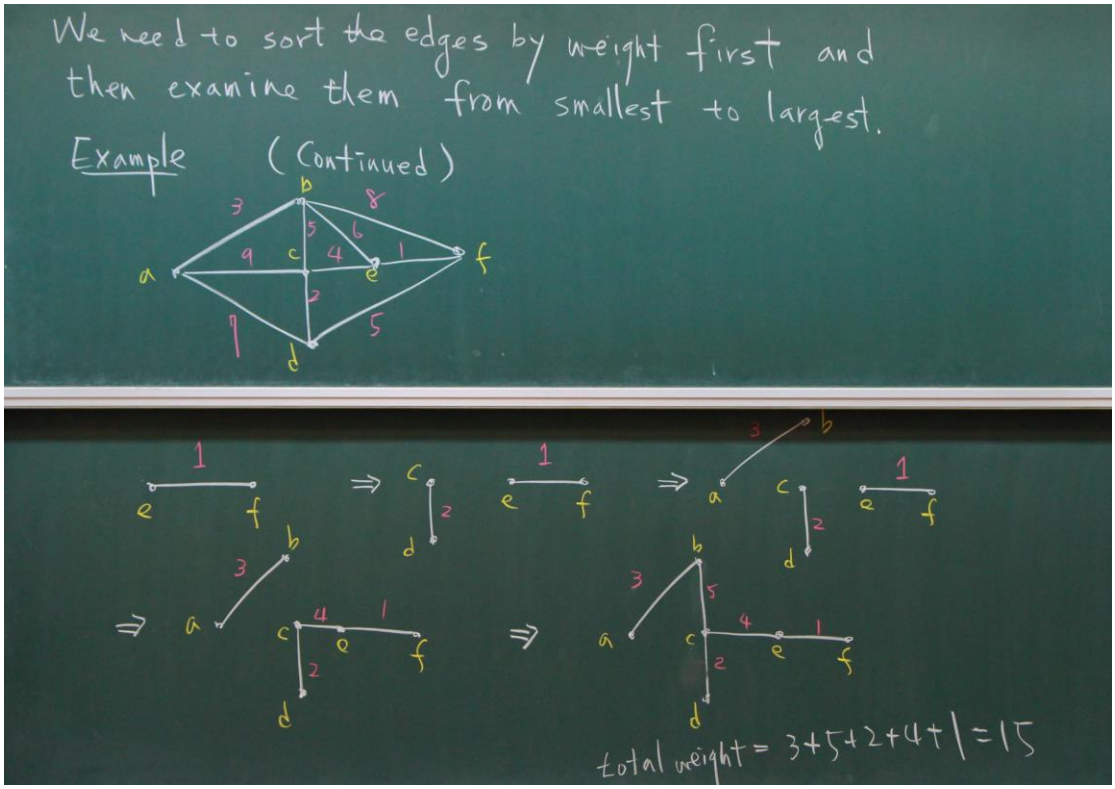
Example (Continued)





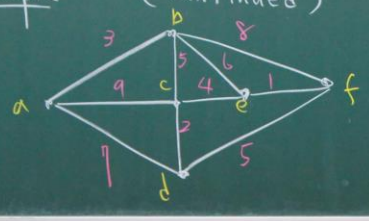
We need to sort the edges by weight first and then examine them from smallest to largest.

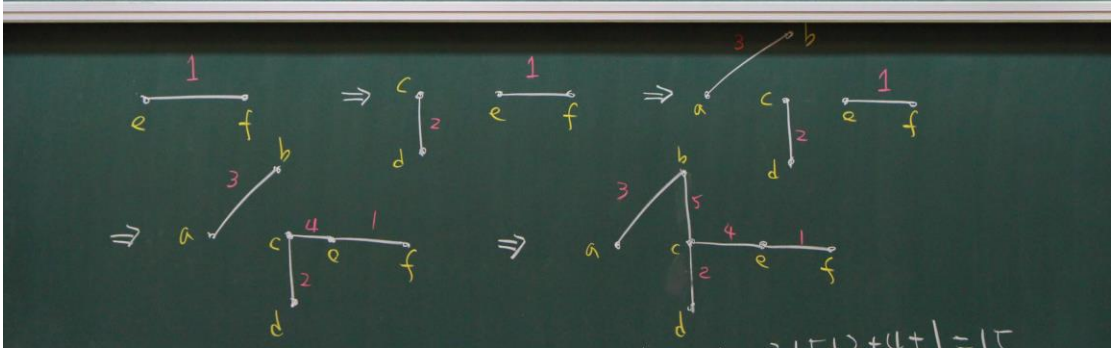
Example (Continued)



We need to sort the edges by weight first and then examine them from smallest to largest.

Example (Continued)





Total weight = 3+5+2+4+1 = 15

Proof of Correctness of Kruskal's Algorithm

The output from Kruskal's algorithm is a graph T, which is a subgraph of G, with n-1 edges and without cycles. We first show that T is connected. Suppose that T has more than one component. Since T has no cycles, each component of T is a tree which has one more vertex than edge.

Proof of Correctness of Kruskal's Algorithm

The output from Kruskal's algorithm is a graph T, which is a subgraph of G, with n-1 edges and without cycles. We first show that T is connected. Suppose that T has more than one component. Since T has no cycles, each component of T is a tree which has one more vertex than edge.

As T has n-1 edges, the number of vertices would be larger than n, contradicting the fact that T is a subgraph of G. Hence T has only one component, thus connected. Consequently, T is a tree with n-1 edges and n vertices, hence a spanning tree of G.

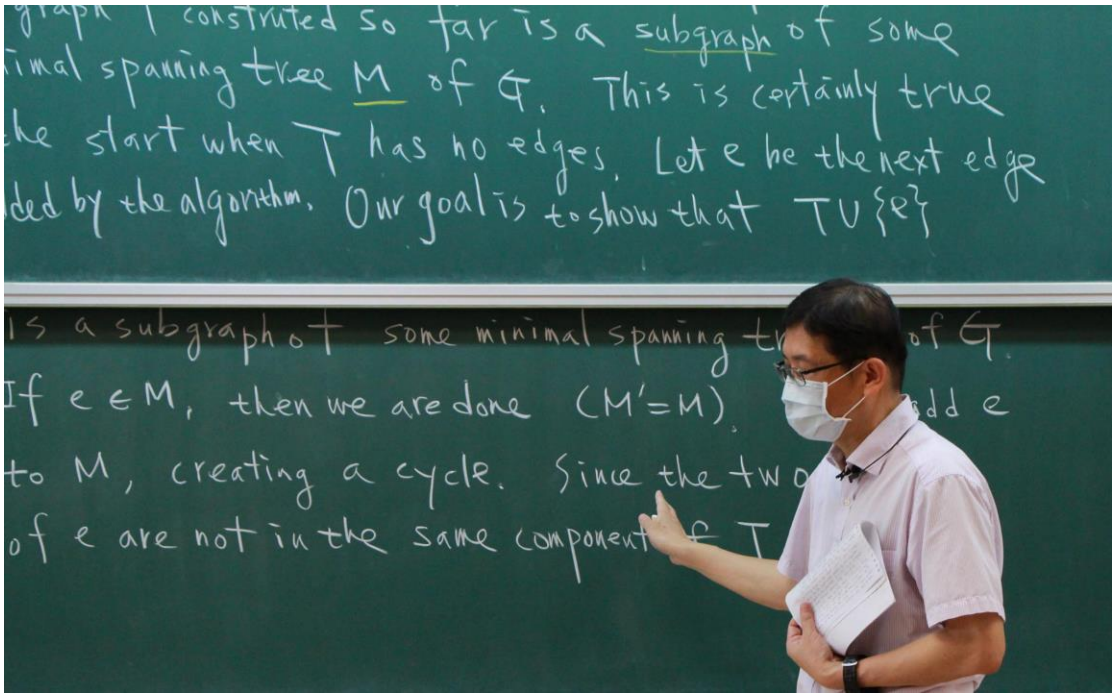
Proof of Correctness of Kruskal's Algorithm

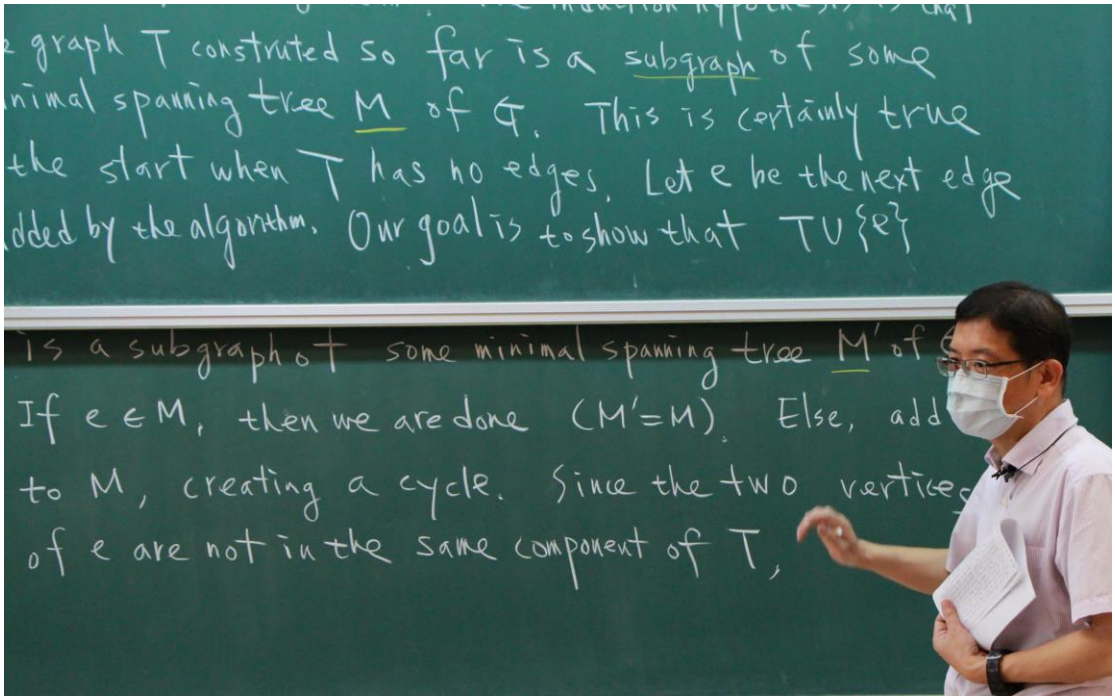
The output from Kruskal's algorithm is a graph T, which is a subgraph of G, with n-1 edges and without cycles. We first show that T is connected. Suppose that T has more than one component. Since T has no cycles, each component of T is a tree which has one more vertex than edge.

As T has n-1 edges, the number of vertices would be larger than n, contradicting the fact that T is a subgraph of G. Hence T has only one component, thus connected. Consequently, T is a tree with n-1 edges and n vertices, hence a spanning tree of G.

We can then use a similar induction argument to the one used for Prim's algorithm. The induction hypothesis is that the graph T constructed so far is a subgraph of some minimal spanning tree M of G. This is certainly true at the start when T has no edges. Let e be the next edge added by the algorithm. Our goal is to show that $T \cup \{e\}$

We can then use a similar induction argument to the one used for Prim's algorithm. The induction hypothesis is that the graph T constructed so far is a subgraph of some minimal spanning tree M of G. This is certainly true at the start when T has no edges. Let e be the next edge added by the algorithm. Our goal is to show that T∪{e}



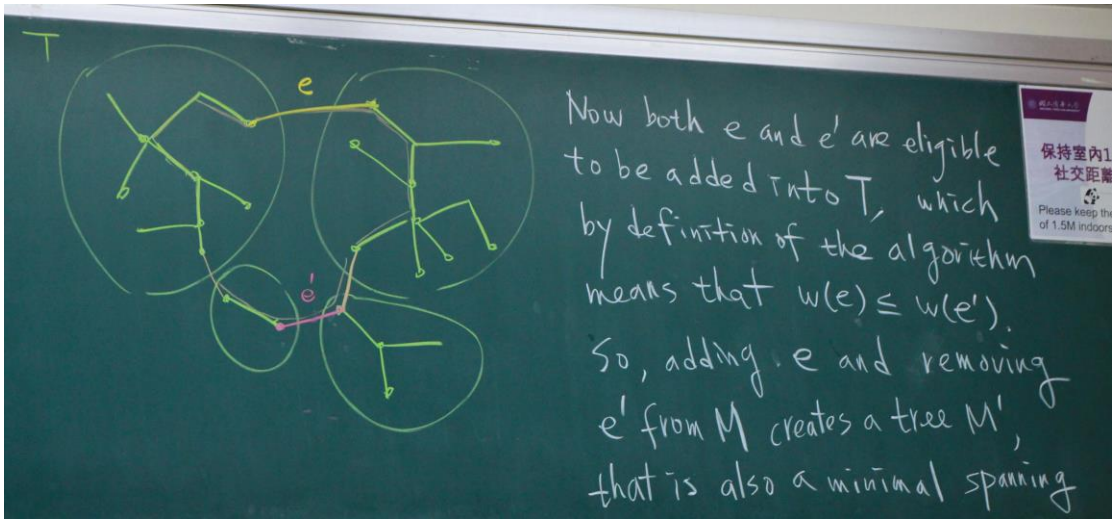


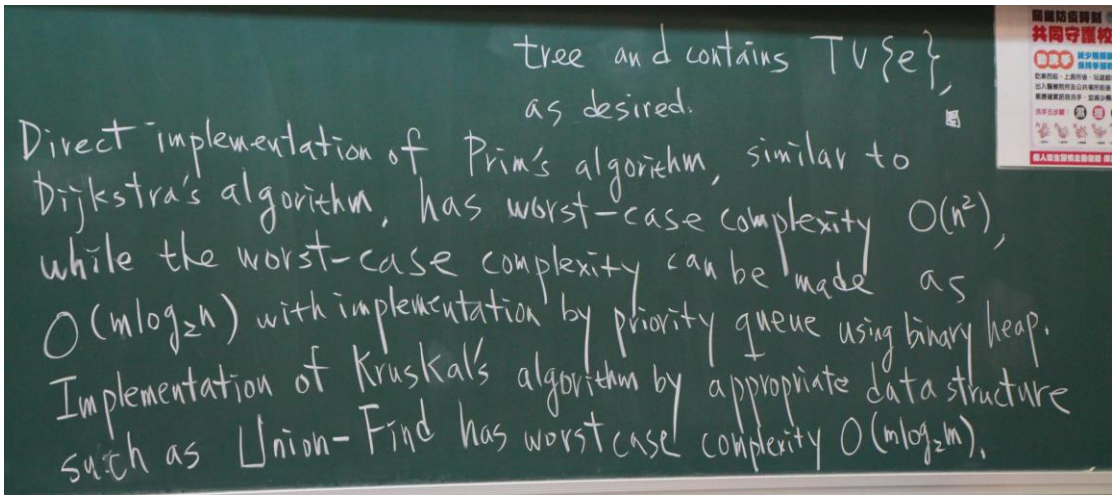
is a subgraph of some minimal spanning tree M' of G.
If $e \in M$, then we are done ($M' = M$). Else, add e
to M, creating a cycle. Since the two vertices
of e are not in the same component of T, if we trace
around the cycle we can find another edge $e' \neq e$ with
two vertices also not in the same component of T.

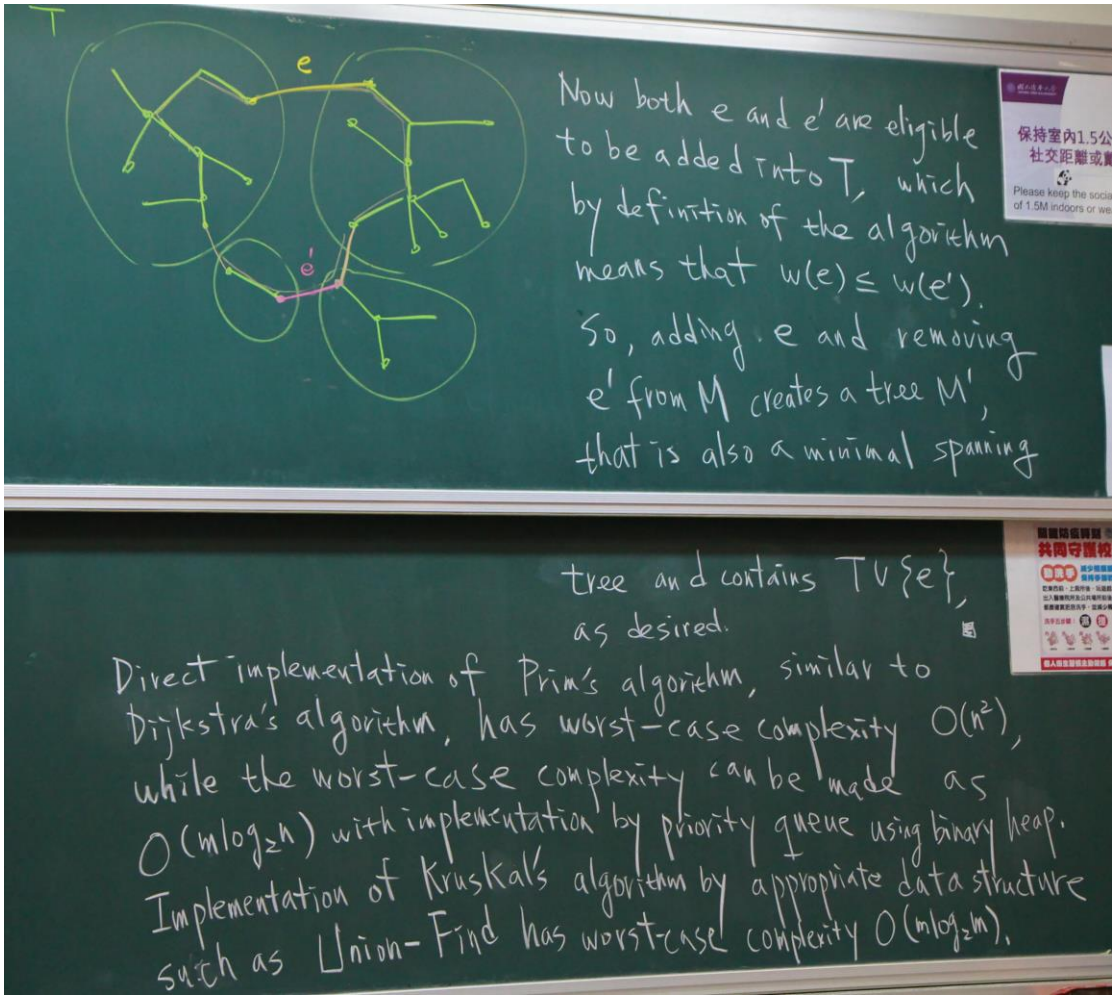
is a subgraph of some minimal spanning tree M' of G.
If $e \in M$, then we are done ($M' = M$). Else, add e
to M, creating a cycle. Since the two vertices
of e are not in the same component of T, if we trace
around the cycle we can find another edge $e' \neq e$ with
two vertices also not in the same component of T.

We can then use a similar induction argument to the one
used for Prim's algorithm. The induction hypothesis is that
the graph T constructed so far is a subgraph of some
minimal spanning tree M of G. This is certainly true
at the start when T has no edges. Let e be the next edge
added by the algorithm. Our goal is to show that $T \cup \{e\}$

is a subgraph of some minimal spanning tree M' of G.
If $e \in M$, then we are done ($M' = M$). Else, add e
to M, creating a cycle. Since the two vertices
of e are not in the same component of T, if we trace
around the cycle we can find another edge $e' \neq e$ with
two vertices also not in the same component of T.


T
e
e'
Now both e and e' are eligible to be added into T, which by definition of the algorithm means that w(e) ≤ w(e'). So, adding e and removing e' from M creates a tree M', that is also a minimal spanning


tree and contains T∪{e}, as desired.
Direct implementation of Prim's algorithm, similar to Dijkstra's algorithm, has worst-case complexity $O(n^2)$, while the worst-case complexity can be made as $O(m\log_2 n)$ with implementation by priority queue using binary heap.
Implementation of Kruskal's algorithm by appropriate data structure such as Union-Find has worst case complexity $O(m\log_2 m)$.


T
e
e'
Now both e and e' are eligible to be added into T, which by definition of the algorithm means that w(e) ≤ w(e'). So, adding e and removing e' from M creates a tree M', that is also a minimal spanning tree and contains T ∪ {e}, as desired.
Direct implementation of Prim's algorithm, similar to Dijkstra's algorithm, has worst-case complexity O(n²), while the worst-case complexity can be made as O(m log₂ n) with implementation by priority queue using binary heap. Implementation of Kruskal's algorithm by appropriate data structure such as Union-Find has worst-case complexity O(m log₂ m).