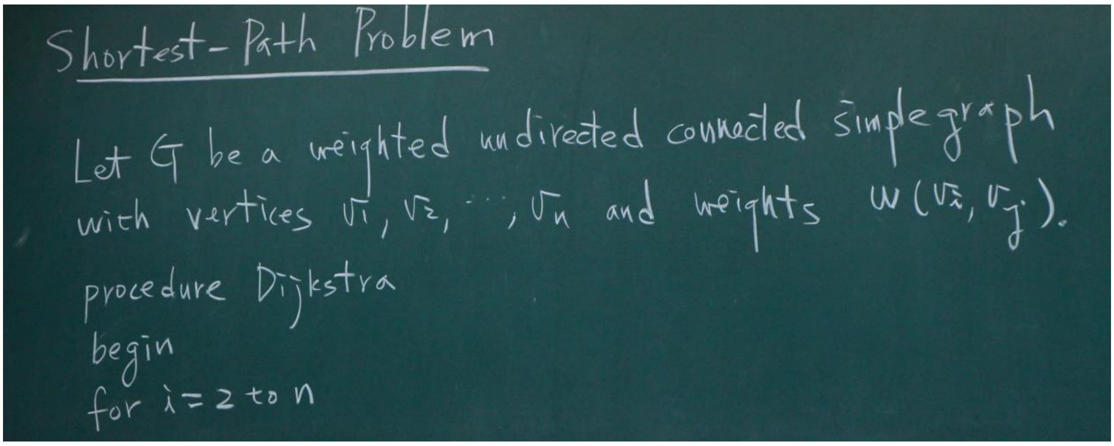
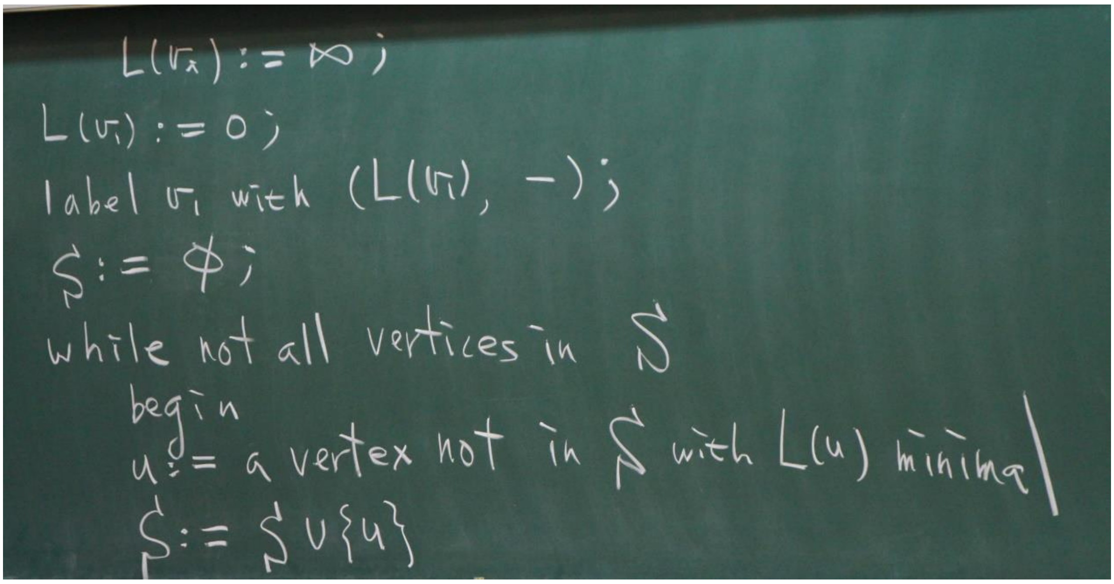
【10920 趙啟超教授離散數學 / 第 27 堂版書】





Shortest-Path Problem

Let G be a weighted undirected connected simple graph with vertices $v_1, v_2, \cdots, v_n$ and weights $w(v_i, v_j)$.

procedure Dijkstra
begin
for $i = 2$ to n
  $L(v_i) := \infty$;
$L(v_1) := 0$;
label v_1 with $(L(v_1), -)$;
$S := \emptyset$;
while not all vertices in S
  begin
  $u :=$ a vertex not in S with $L(u)$ minimal
  $S := S \cup \{u\}$
  for all vertices v not in S
    if $L(u) + w(u,v) < L(v)$ then
      $L(v) := L(u) + w(u,v)$ and label vertex v with $(L(v), u)$
  end
end

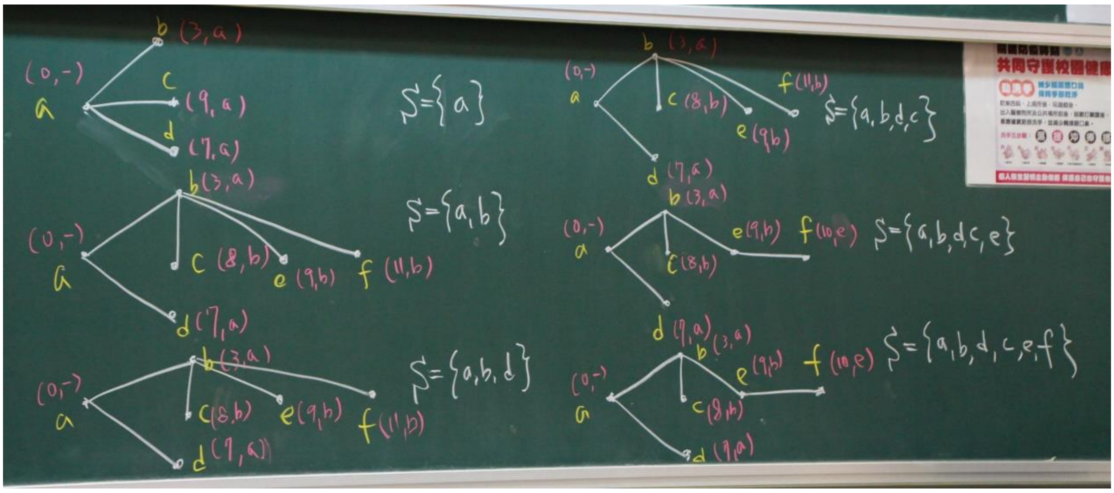
When Dijkstra's algorithm ends, the label $(L(v_i), u)$ of vertex v_i means the following:

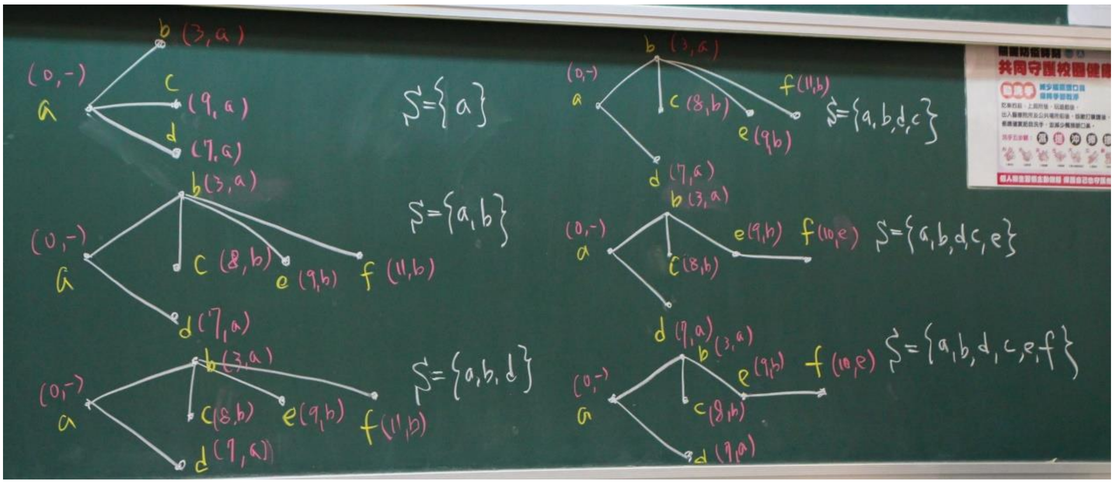
1. $L(v_i)$ is the weight of the shortest path from v_1 to v_i.
2. u is the predecessor vertex of v_i along the shortest path from v_1 to v_i.

When Dijkstra's algorithm ends, the label $(L(v_i), u)$ of vertex v_i means the following:

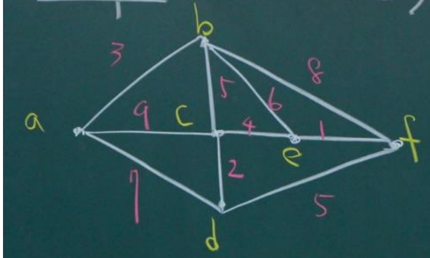
1. $L(v_i)$ is the weight of the shortest path from v_1 to v_i.
2. u is the predecessor vertex of v_i along the shortest path from v_1 to v_i.

```
    for all vertices v not in S
        if L(u) + w(u,v) < L(v) then
            L(v) := L(u) + w(u,v) and label vertex v with (L(v),u)
    end
end
```





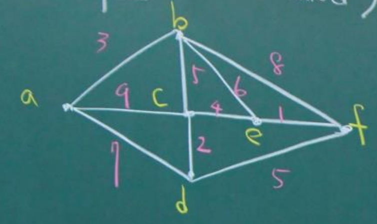
Example (Continued)



$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline a & \infty & 3 & 9 & 7 & \infty & \infty \\ b & 3 & \infty & 5 & \infty & 6 & 8 \\ c & 9 & 5 & \infty & 2 & 4 & \infty \\ d & 7 & \infty & 2 & \infty & \infty & 5 \\ e & \infty & 6 & 4 & \infty & \infty & 1 \\ f & \infty & 8 & \infty & 5 & 1 & \infty \end{array}$$

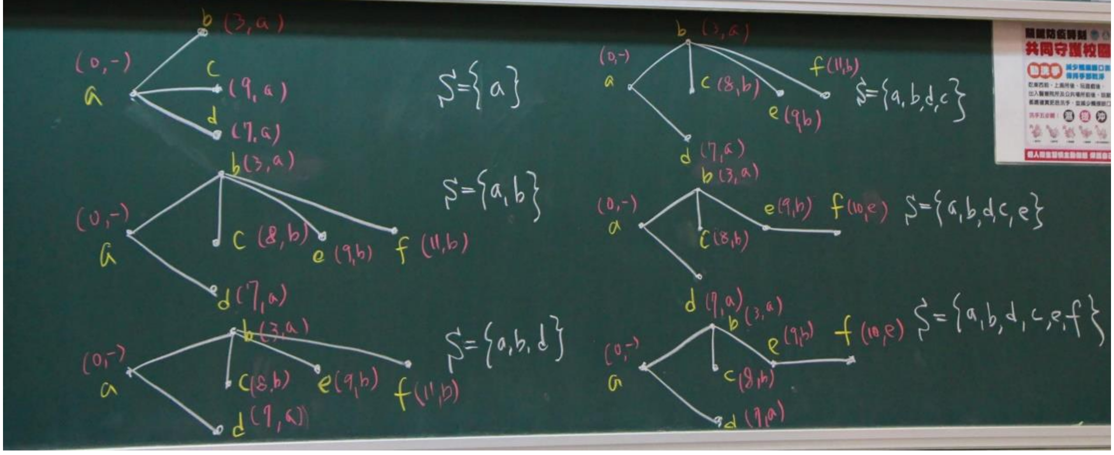
The shortest path from a to f is abef with weight 10. The shortest path from a to c is abc with weight 8.

Example (Continued)



$$
\begin{array}{c}
\\ a \\ b \\ c \\ d \\ e \\ f
\end{array}
\begin{array}{c}
\begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\begin{bmatrix}
\infty & 3 & 9 & 7 & \infty & \infty \\
3 & \infty & 5 & \infty & 6 & 8 \\
9 & 5 & \infty & 2 & 4 & \infty \\
7 & \infty & 2 & \infty & \infty & 5 \\
\infty & 6 & 4 & \infty & \infty & 1 \\
\infty & 8 & \infty & 5 & 1 & \infty
\end{bmatrix}
\end{array}
$$

The shortest path from a to f is abef with weight 10. The shortest path from a to c is abc with weight 8.



Consequently, the total number of comparisons is at most $\sum_{i=1}^{n-1} 2(n-i) = n^2-n$ and the total number of additions is at most $\sum_{i=1}^{n-1}(n-i) = \frac{n^2-n}{2}$. Hence the worst-case complexity is $O(n^2)$ for direct implementation.

For other implementation by a data structure called priority queue using binary heap, the worst-case complexity can be made as $O(m \log_2 n)$ for a weighted graph $G=(V,E)$ with $|V|=n$ and $|E|=m$.

Consequently, the total number of comparisons is at most $\sum_{i=1}^{n-1} 2(n-i) = n^2-n$ and the total number of additions is at most $\sum_{i=1}^{n-1} (n-i) = \frac{n^2-n}{2}$. Hence the worst-case complexity is $O(n^2)$ for direct implementation.

For other implementation by a data structure called priority queue using binary heap, the worst-case complexity can be made as $O(m \log_2 n)$ for a weighted graph $G=(V,E)$ with $|V|=n$ and $|E|=m$.

Consequently, the total number of comparisons is at most $\sum_{i=1}^{n-1} 2(n-i) = n^2 - n$ and the total number of additions is at most $\sum_{i=1}^{n-1} (n-i) = \frac{n^2-n}{2}$. Hence the worst-case complexity is $O(n^2)$ for direct implementation.

For other implementation by a data structure called priority quene using binary heap, the worst-case complexity can be made as $O(m \log_2 n)$ for a weighted graph $G=(V,E)$ with $|V|=n$ and $|E|=m$.

$L(v) := L(u) + w(u,v)$ ...

end

end

In Dijkstra's algorithm, there are at most $n-1$ iterations.

In the first iteration, at most $n-1$ comparisons are used to find the minimal $L(u)$, and another $n-1$ comparisons and $n-1$ additions at most are used to update the labels.

In general, in the ith iteration, $1 \le i \le n-1$, at most $n-i$ comparisons are used to find the minimal $L(u)$, and another $n-i$ comparisons and $n-i$ additions at most are used to update the labels.

Minimal Spanning Trees

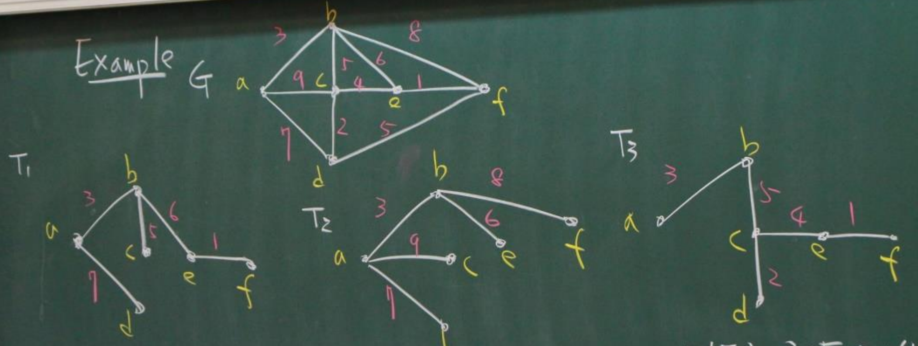
Consider a weighted undirected connected simple graph $G = (V, E)$ with $|V| = n$ and $|E| = m$.

A minimal spanning tree in G is a spanning tree that has the smallest possible sum of weights of its edges.

Consider a weighted undirected connected simple graph $G = (V, E)$ with $|V| = n$ and $|E| = m$.

A minimal spanning tree in G is a spanning tree that has the smallest possible sum of weights of its edges.

Example



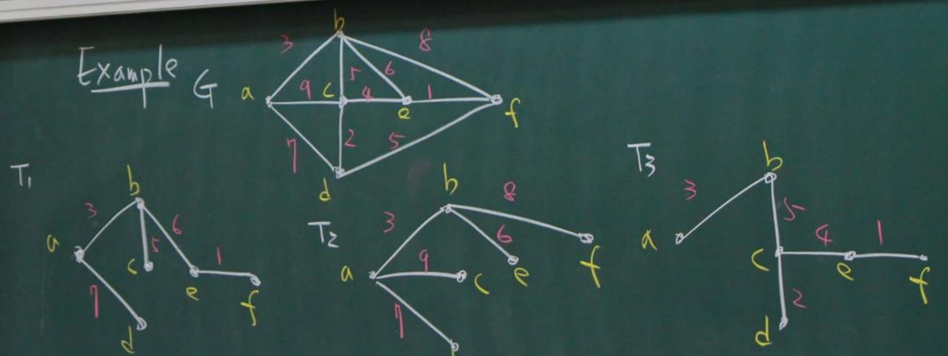
$w(T_1) = 3+5+7+6+1 = 22$ $w(T_2) = 3+9+7+6+8 = 33$ $w(T_3) = 3+5+2+4+1 = 15$

Minimal Spanning Trees

Consider a weighted undirected connected simple graph $G = (V, E)$ with $|V| = n$ and $|E| = m$.

A minimal spanning tree in G is a spanning tree that has the smallest possible sum of weights of its edges.

Example



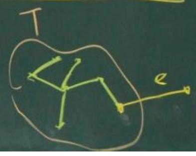
$w(T_1) = 3+5+7+6+1 = 22$ $w(T_2) = 3+9+7+6+8 = 33$ $w(T_3) = 3+5+2+4+1 = 15$

Greedy algorithm: repeatedly makes locally best choices/decisions, ignoring effects in the future.

procedure Prim
begin
T := tree consisting of any single vertex;
for $i = 1$ to $n-1$
    begin
    e := an edge of minimum weight incident to

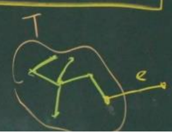
begin
T := tree consisting of any single vertex;
for $i = 1$ to $n-1$
    begin
    e := an edge of minimum weight incident to

            a vertex in T and a vertex not in T
    add e to T
    end
end

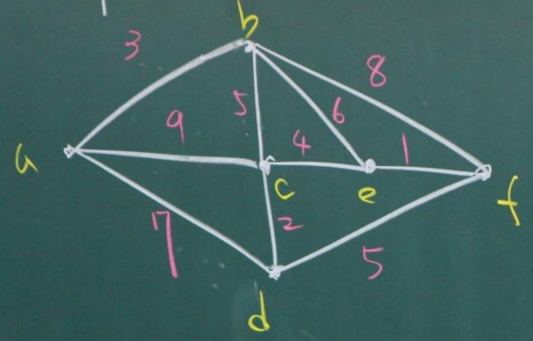


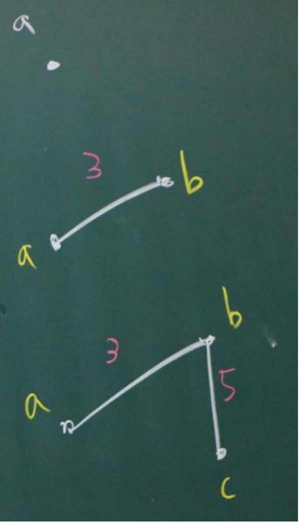
Greedy algorithm: repeatedly makes locally best choices/decisions, ignoring effects in the future.

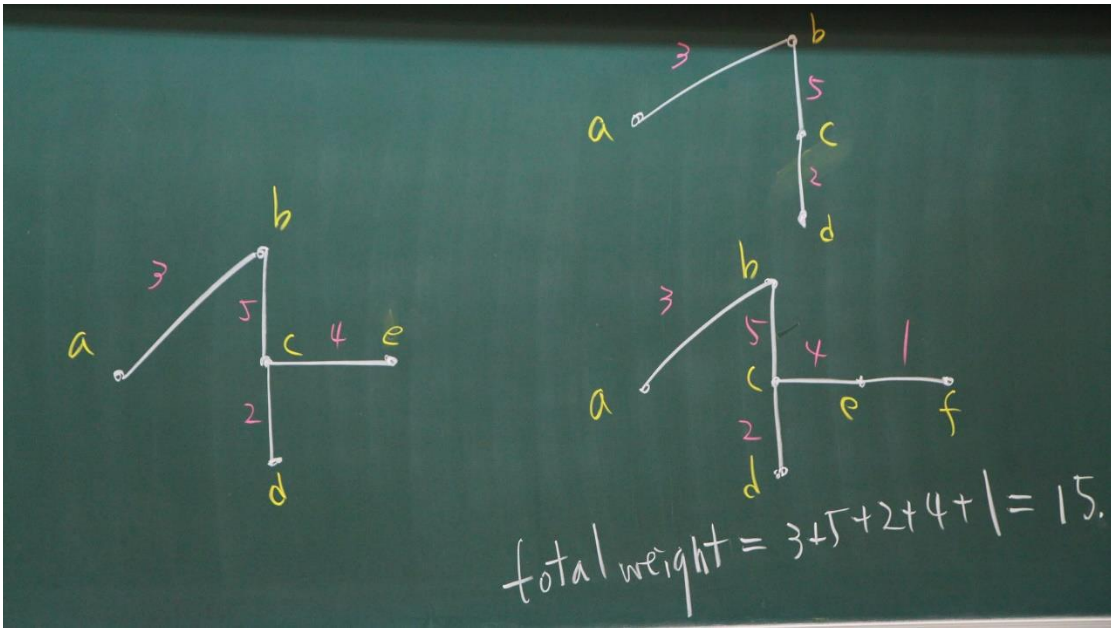
```
procedure Prim
begin
T: = tree consisting of any single vertex;
for i = 1 to n-1
    begin
    e: = an edge of minimum weight incident to
         a vertex in T and a vertex not in T
    add e to T
    end
end
```

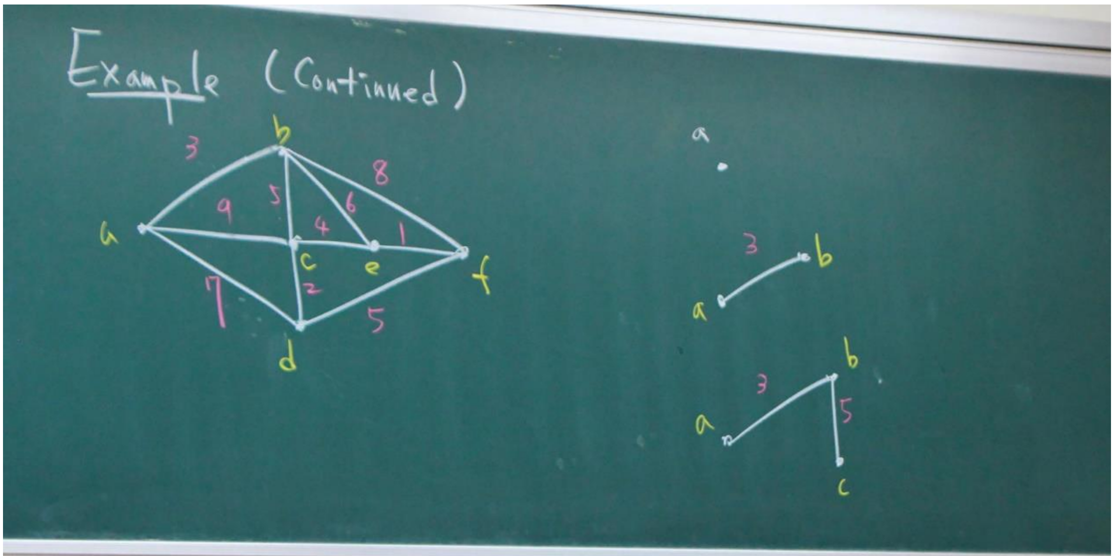


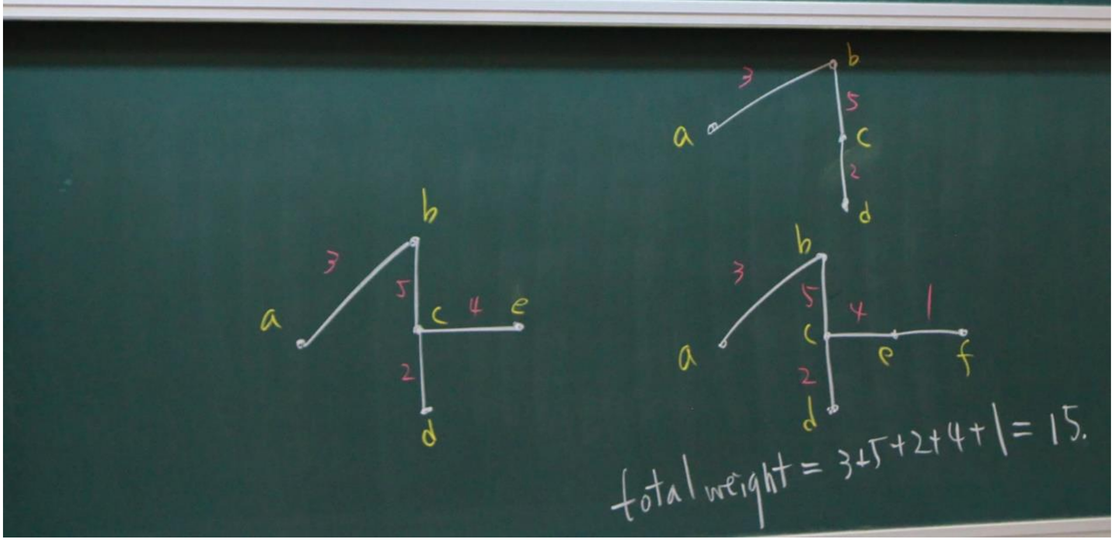
Example (Continued)






a
b
c
d
e
f
3
5
2
4
1
total weight = 3+5+2+4+1 = 15.


Example (Continued)
a
b
c
d
e
f
3
9
5
6
8
4
1
7
2
5


a
b
c
d
e
f
3
5
2
4
1
total weight = 3+5+2+4+1 = 15.

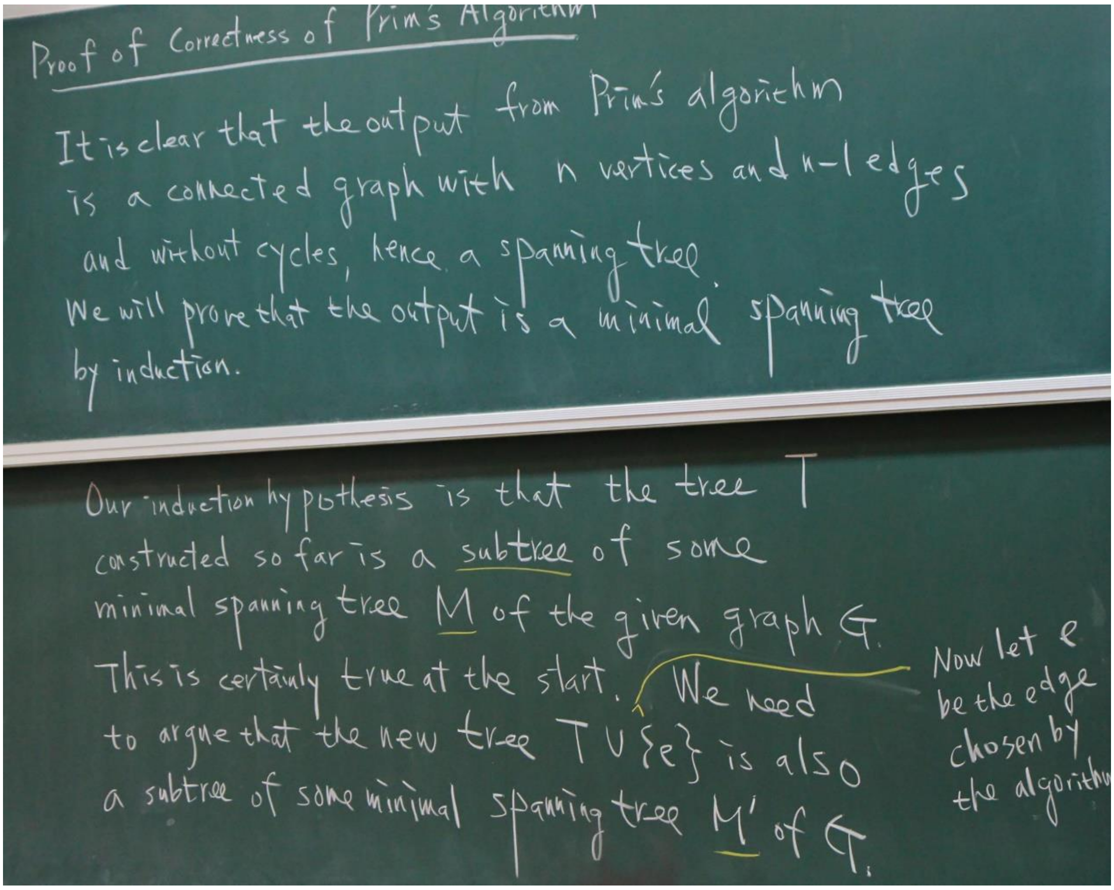
Proof of Correctness of Prim's Algorithm

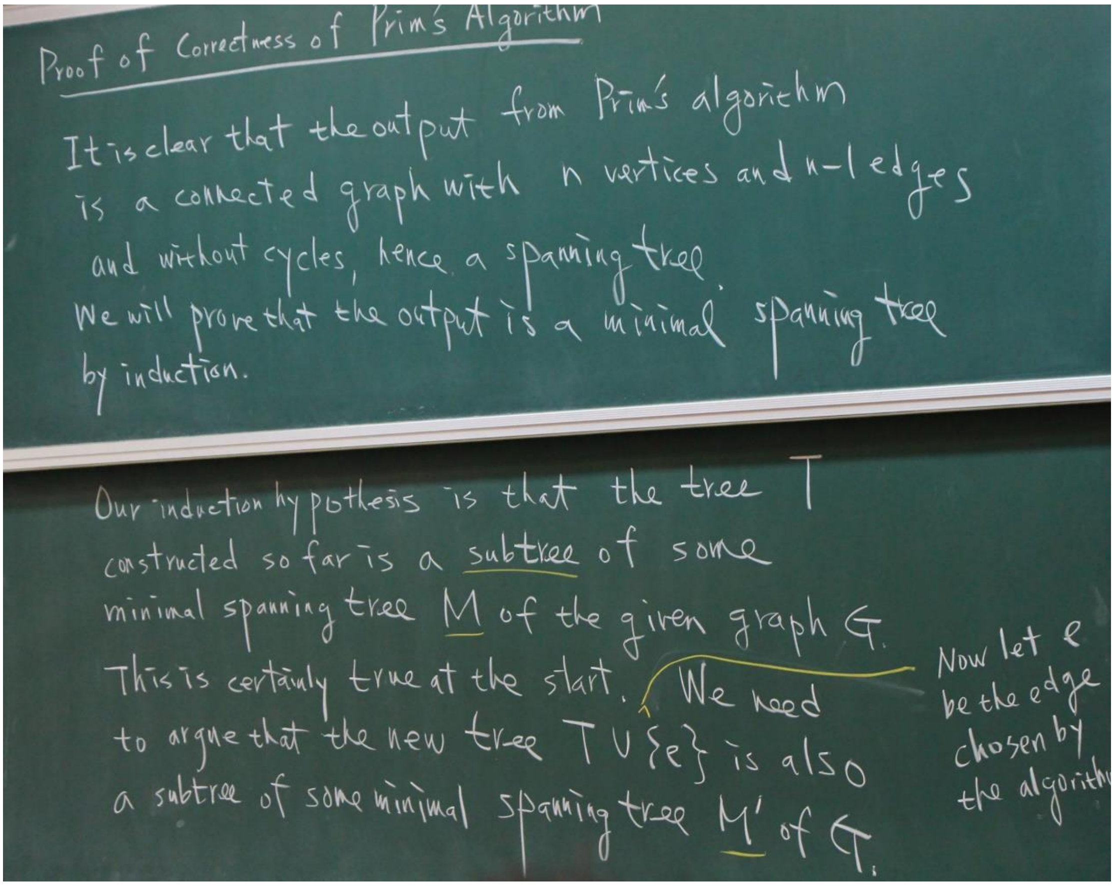
It is clear that the output from Prim's algorithm is a connected graph with n vertices and $n-1$ edges and without cycles, hence a spanning tree.
We will prove that the output is a minimal spanning tree by induction.

and without cycles, hence a spanning tree.
We will prove that the output is a minimal spanning tree by induction.

Our induction hypothesis is that the tree T constructed so far is a subtree of some minimal spanning tree M of the given graph G.
This is certainly true at the start. We need to argue that the new tree $T \cup \{e\}$ is also a subtree of some minimal spanning tree M' of G.

Now let e be the edge chosen by the algorithm.

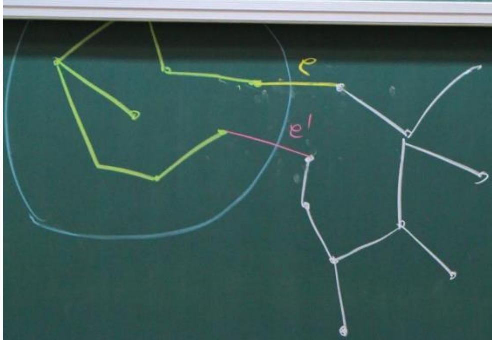

Proof of Correctness of Prim's Algorithm
It is clear that the output from Prim's algorithm
is a connected graph with n vertices and n-1 edges
and without cycles, hence a spanning tree.
We will prove that the output is a minimal spanning tree
by induction.
Our induction hypothesis is that the tree T
constructed so far is a subtree of some
minimal spanning tree M of the given graph G.
This is certainly true at the start. We need
to argue that the new tree T ∪ {e} is also
a subtree of some minimal spanning tree M' of G.
Now let e
be the edge
chosen by
the algorith


Proof of Correctness of Prim's Algorithm
It is clear that the output from Prim's algorithm
is a connected graph with n vertices and n-1 edges
and without cycles, hence a spanning tree.
We will prove that the output is a minimal spanning tree
by induction.
Our induction hypothesis is that the tree T
constructed so far is a subtree of some
minimal spanning tree M of the given graph G.
This is certainly true at the start. We need
to argue that the new tree T ∪ {e} is also
a subtree of some minimal spanning tree M' of G.
Now let e
be the edge
chosen by
the algorith

If $e \in M$ then we are done ($M' = M$).

Else, we argue as follows. Consider adding e to M, which then creates a cycle. Since e has one vertex in T and one vertex not in T, we trace around this cycle and can find another edge $e' \neq e$ with one vertex in T and one vertex not in T.

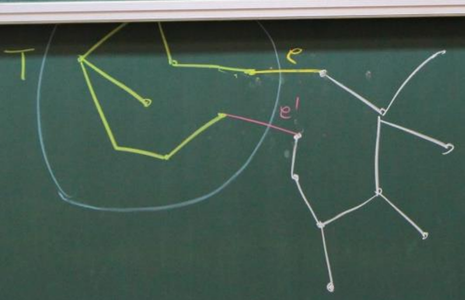
vertex not in T, we trace around this cycle and
find another edge $e' \neq e$ with one vertex in T
one vertex not in T.



We know $w(e') \geq w(e)$ by the definition of the algorithm. So if we add e to M and remove e', we get a new spanning tree M' that has weight not larger than M, thus also a minimal spanning tree, and contains $T \cup \{e\}$, which completes the induction step. ∎

If $e \in M$ then we are done ($M' = M$).

Else, we argue as follows. Consider adding e to M, which then creates a cycle. Since e has one vertex in T and one vertex not in T, we trace around this cycle and can find another edge $e' \neq e$ with one vertex in T and one vertex not in T.



We know $w(e') \geq w(e)$ by the definition of the algorithm. So if we add e to M and remove e', we get a new spanning tree M' that has weight not larger than M, thus also a minimal spanning tree, and contains $T \cup \{e\}$, which completes the induction step. ∎