Greedy Algorithms 2

The Minimum Spanning Tree Problem

The Minimum Spanning Tree Problem

One of the most basic problems to do with graphs:

> *What is the "cheapest" way to interconnect objects in a network?*

Model the situation as a graph where:

- Assume edges have weights that give "cost" of a connection
- Being interconnected means that there must be at least one path between every pair of objects in network

Formulating the MST Problem

Given a graph $G = (V, E)$ and edge weight function w

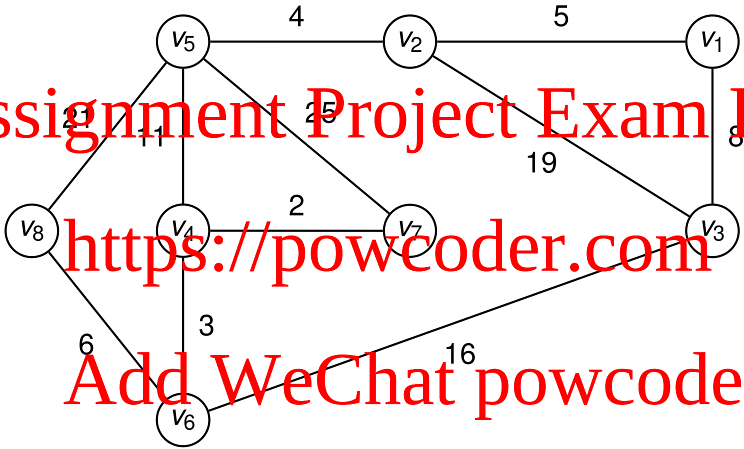
Find a subset E' of E such that:

1. $G' = (V, E')$ is **connected**
2. The total of the weights of edges in E' is minimised

Two things to note:

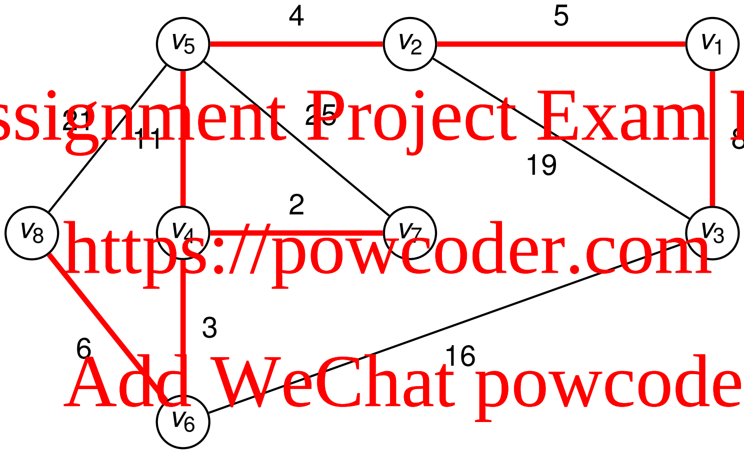
- G' must be a tree
  — otherwise leave edge(s) out to reduce weight
- A path from u to v in G' is not necessarily shortest paths in G

MST Example



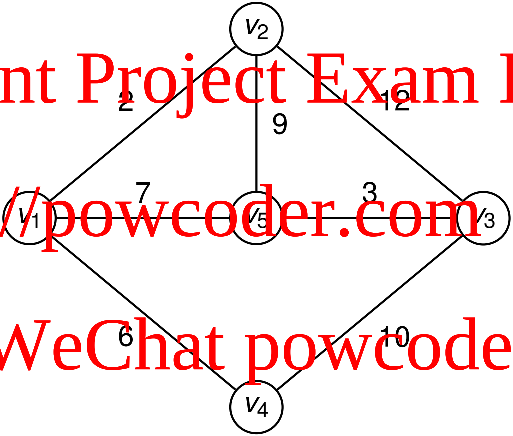
What is a minimum spanning tree for this weighted graph ?

MST Example



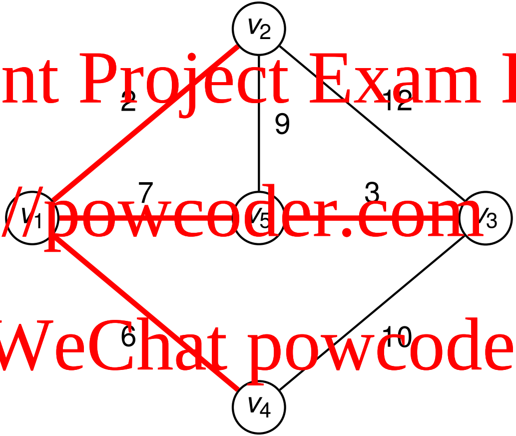
Consider the path from v_3 to v_6 — length is more than 16

MST Example for You



Indicate nodes that are in a minimum spanning tree for this weighted graph?

MST Example for You



Indicate nodes that are in a minimum spanning tree for this weighted graph?

Solving the MST Problem

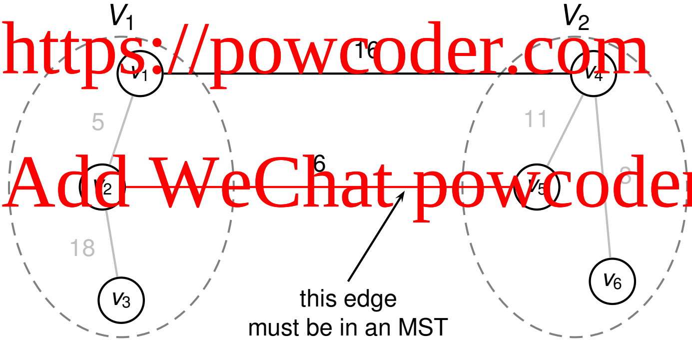
Turns out to be easy to solve using the greedy approach

- Consider edges in order of increasing weight and include in E' as long as no cycle created
- Grow tree from arbitrary starting point, repeatedly connecting in vertices that are closest to what has been built so far
- Consider edges in decreasing order of weight and exclude from G unless it disconnects the graph

Greedy approaches work because of the so-called **Cut Property**

Cut Property

Given a graph $G = (V, E)$ and edge weight function w, if V is split into two disjoint sets V_1 and V_2 then the least weighted edge $\{u, v\}$ with $u \in V_1$ and $v \in V_2$ is contained in every minimum spanning tree of G.



Proof of Cut Property

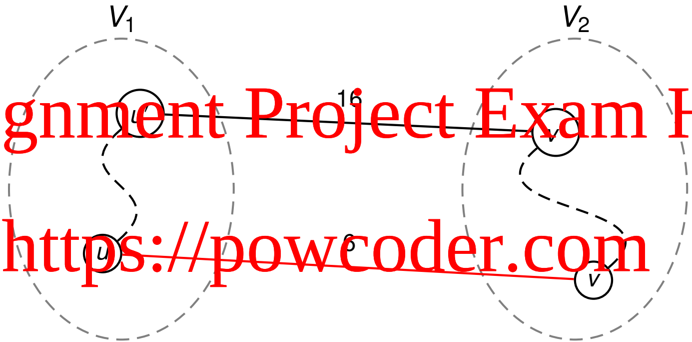
Proof by contradiction (assumes all edge weights distinct):

- Suppose that sets V_1 and V_2 and edge $\{u, v\}$ are as in the proposition
- $\{u, v\}$ is **not** in the MST $G' = (V, E')$ for the graph

- Consider path p in G' from u to v
- Path p must cross from V_1 to V_2
- Let $\{u', v'\}$ be the edge that does this

Time for a picture

Proof of Cut Property (cont.)

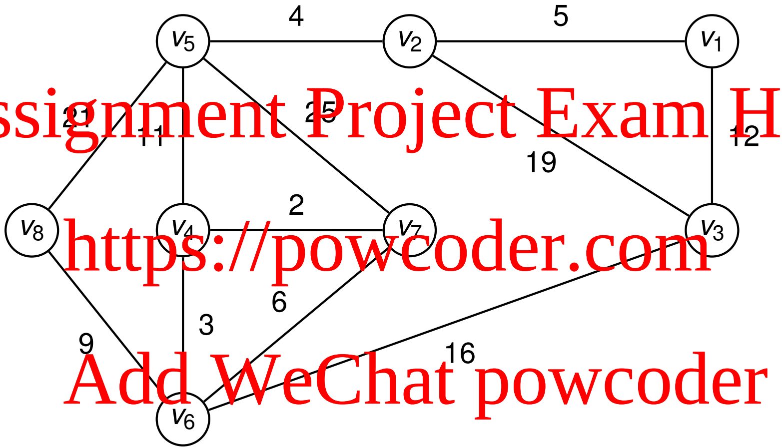


- Any pair of vertices connected via $\{u', v'\}$ still connected via $\{u, v\}$
- We are reducing weight of E' by substituting $\{u', v'\}$ by $\{u, v\}$
- Contradiction — G' couldn't have been a MST

Kruskal's MST Algorithm

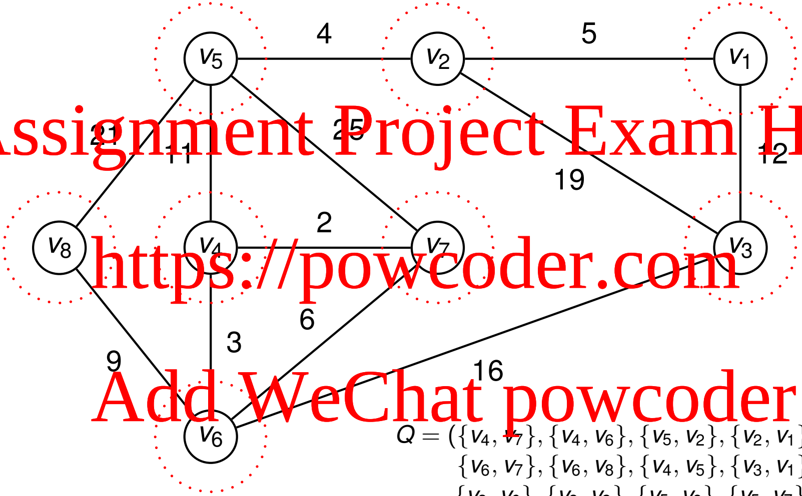
Kruskal(G, w) :
    let Q be the edges in E ordered by increasing weight
    initialise E' *to be the empty set*
    for each vertex $v \in V$
        let $C(v)$ *be the singleton set containing v*
    while $|E'| < |V| - 1$
        remove edge $\{u, v\}$ *from Q*
        if $C(u) \neq C(v)$ *then*
            include $\{u, v\}$ *in* E'
            $C(u) \leftarrow C(v) \leftarrow C(u) \cup C(v)$
    return E'

Illustration of Kruskal's Algorithm



Create Q and $C(v)$ for each $v \in V$

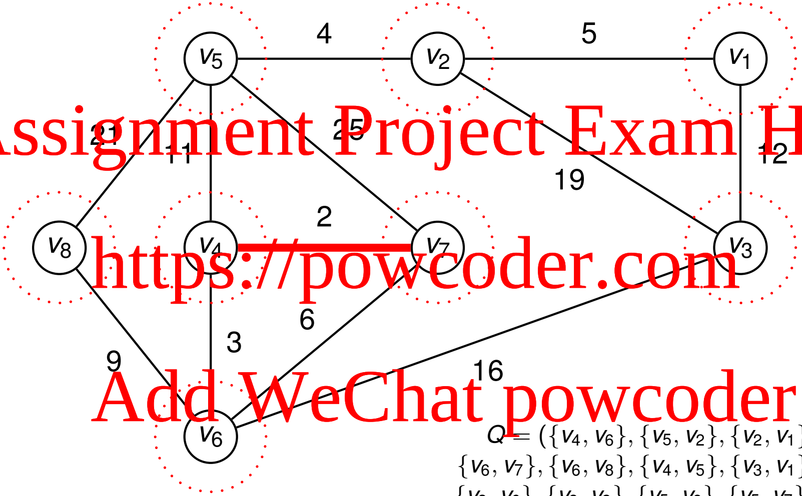
Illustration of Kruskal's Algorithm



$$Q = (\{v_4, v_7\}, \{v_4, v_6\}, \{v_5, v_2\}, \{v_2, v_1\},$$
$$\{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\},$$
$$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$$

Remove edge $\{v_4, v_7\}$ from Q

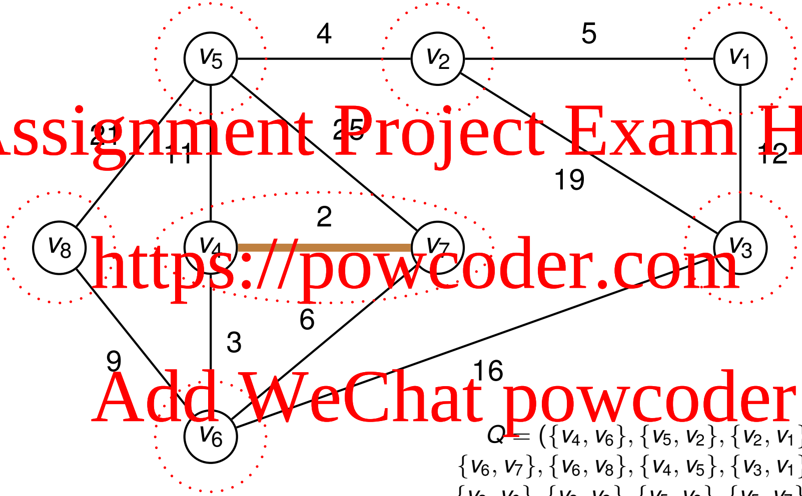
Illustration of Kruskal's Algorithm



$Q = (\{v_4, v_6\}, \{v_5, v_2\}, \{v_2, v_1\},$
$\{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

$C(v_4) \neq C(v_7)$ so add to E' and combine clusters

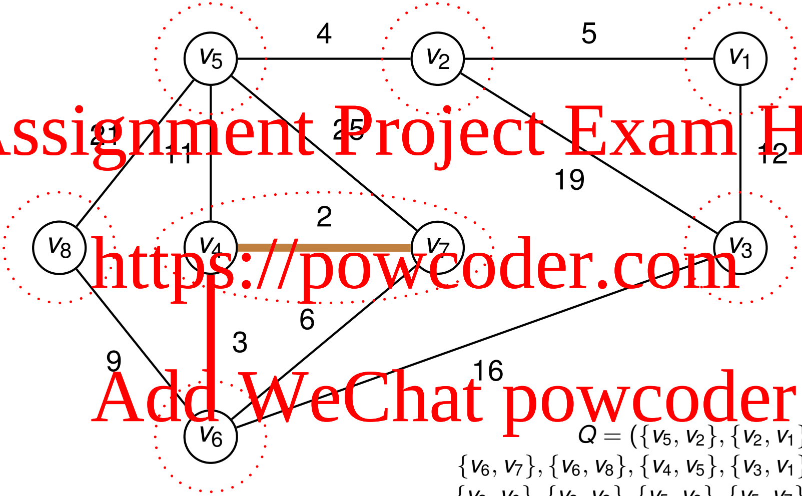
Illustration of Kruskal's Algorithm



$Q = (\{v_4, v_6\}, \{v_5, v_2\}, \{v_2, v_1\},$
$\{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

Remove edge $\{v_4, v_6\}$ from Q

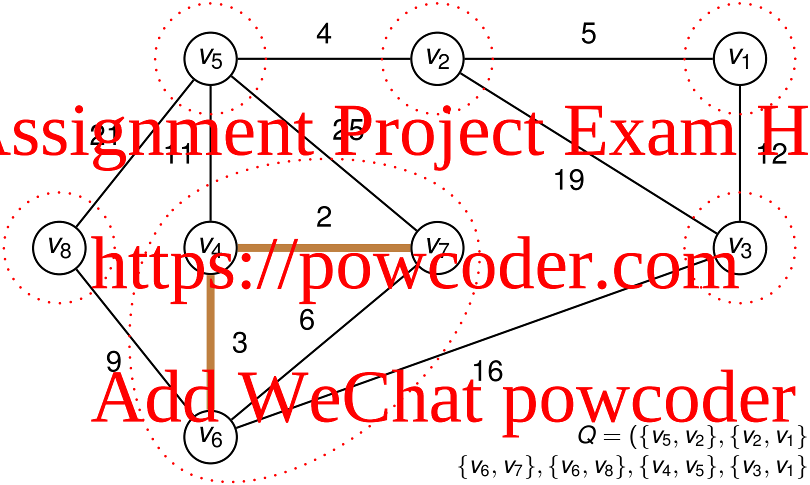
Illustration of Kruskal's Algorithm



$Q = (\{v_5, v_2\}, \{v_2, v_1\}, \{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\}, \{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

$C(v_4) \neq C(v_6)$ so add to E' and combine clusters

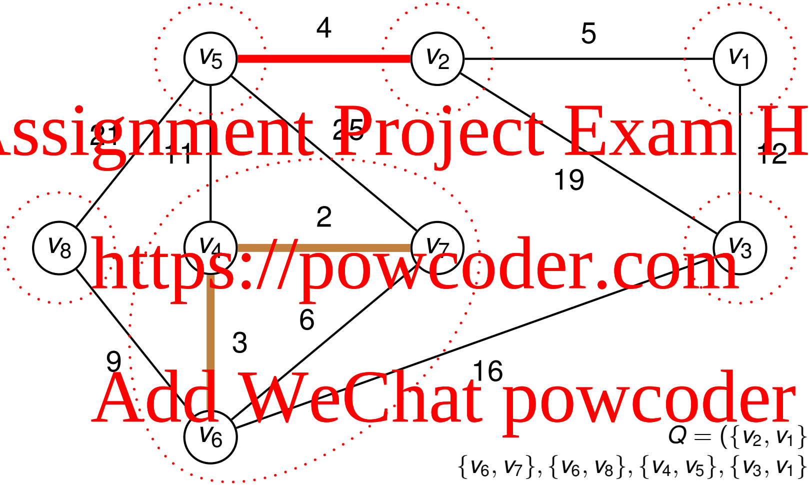
Illustration of Kruskal's Algorithm



$Q = (\{v_5, v_2\}, \{v_2, v_1\},$
$\{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

Remove edge $\{v_5, v_2\}$ from Q

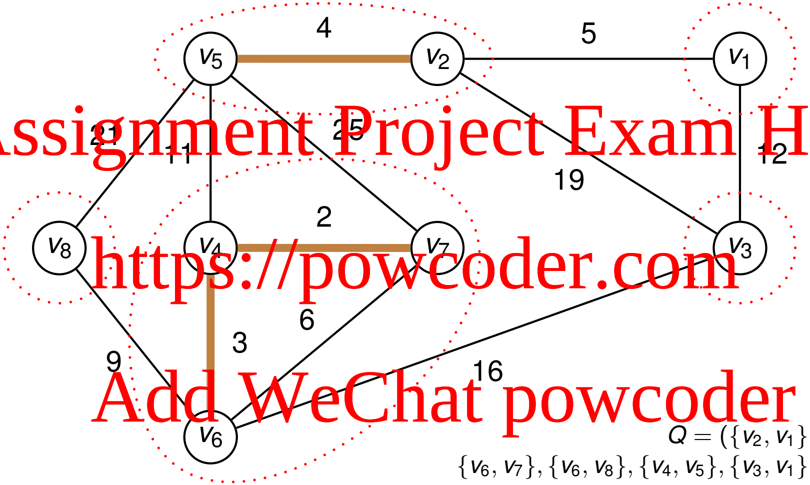
Illustration of Kruskal's Algorithm



$Q = (\{v_2, v_1\},$
$\{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

$C(v_2) \neq C(v_5)$ so add to E' and combine clusters

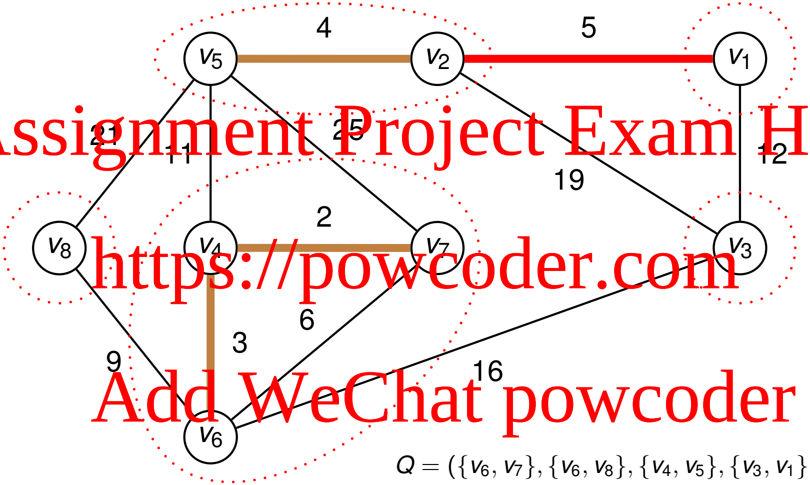
Illustration of Kruskal’s Algorithm



$Q = (\{v_2, v_1\},$
$\{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

Remove edge $\{v_2, v_1\}$ from Q

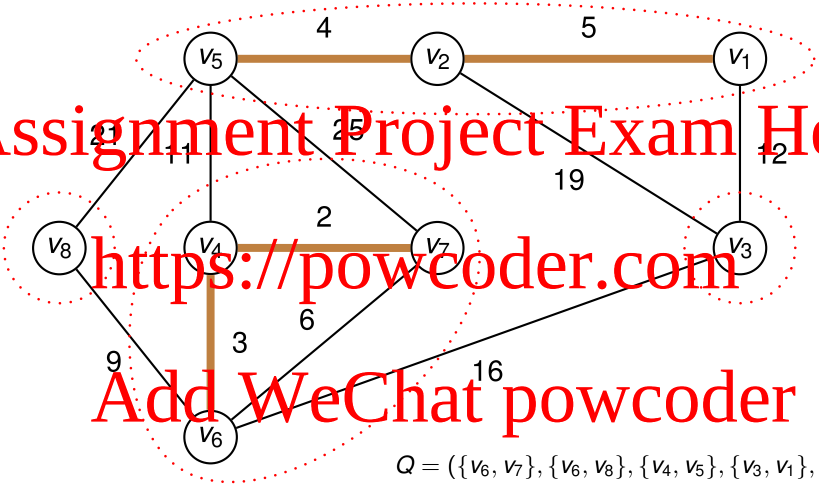
Illustration of Kruskal's Algorithm



$$Q = (\{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\}, \{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$$

$C(v_2) \neq C(v_1)$ so add to E' and combine clusters

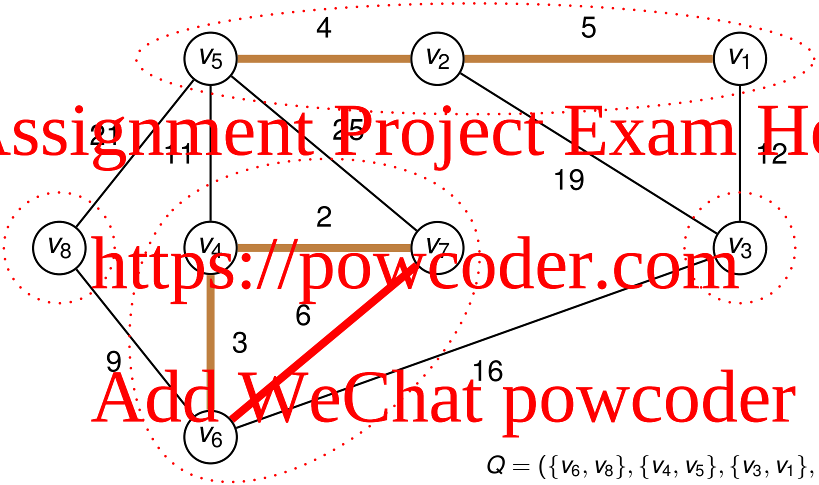
Illustration of Kruskal's Algorithm



$$Q = (\{v_6, v_7\}, \{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\}, \{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$$

Remove edge $\{v_6, v_7\}$ from Q

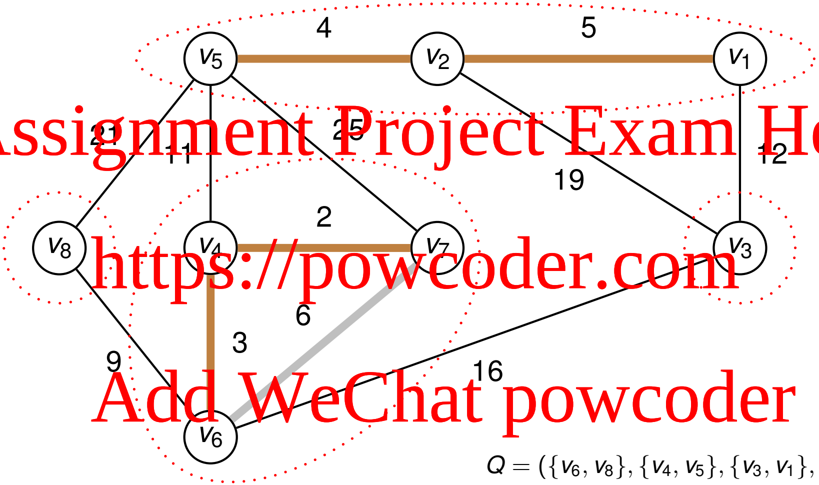
Illustration of Kruskal's Algorithm



$Q = (\{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\}, \{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

$C(v_6) = C(v_7)$ so don't add to E'

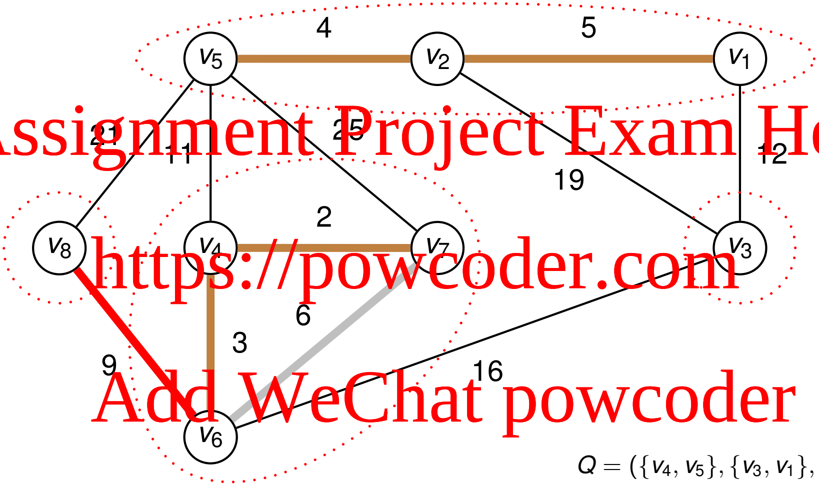
Illustration of Kruskal's Algorithm



$Q = (\{v_6, v_8\}, \{v_4, v_5\}, \{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

Remove edge $\{v_6, v_7\}$ from Q

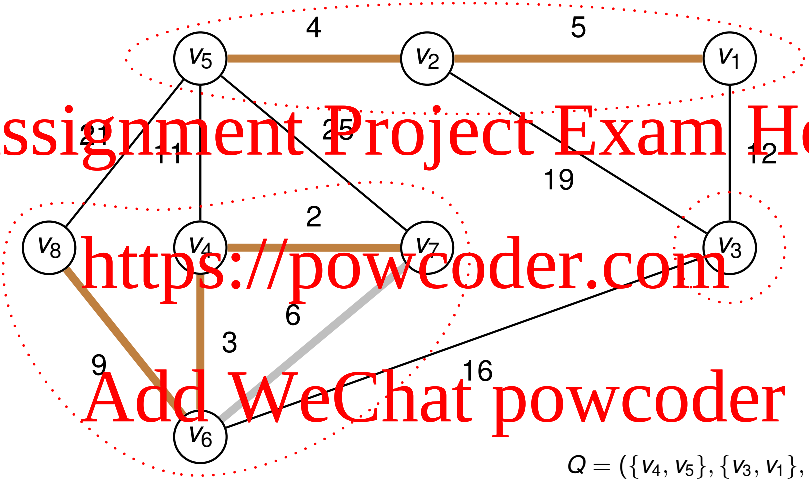
Illustration of Kruskal’s Algorithm



$Q = (\{v_4, v_5\}, \{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

$C(v_6) \neq C(v_8)$ so add to E' and combine clusters

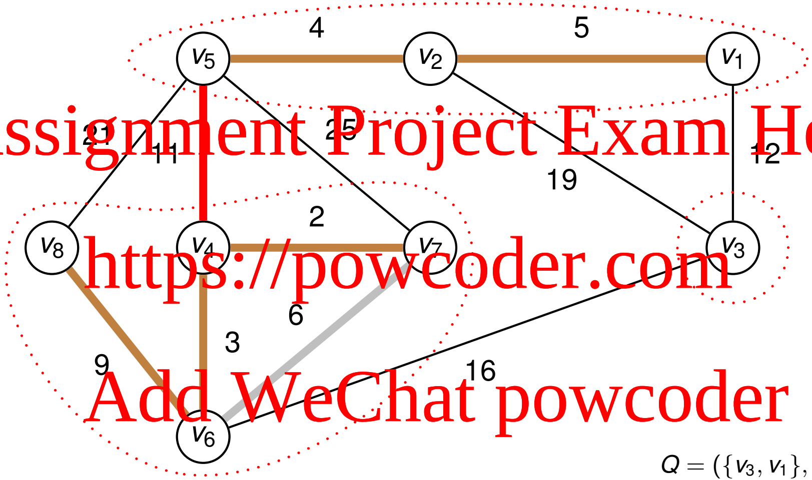
Illustration of Kruskal's Algorithm



$Q = (\{v_4, v_5\}, \{v_3, v_1\}, \{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

Remove edge $\{v_4, v_5\}$ from Q

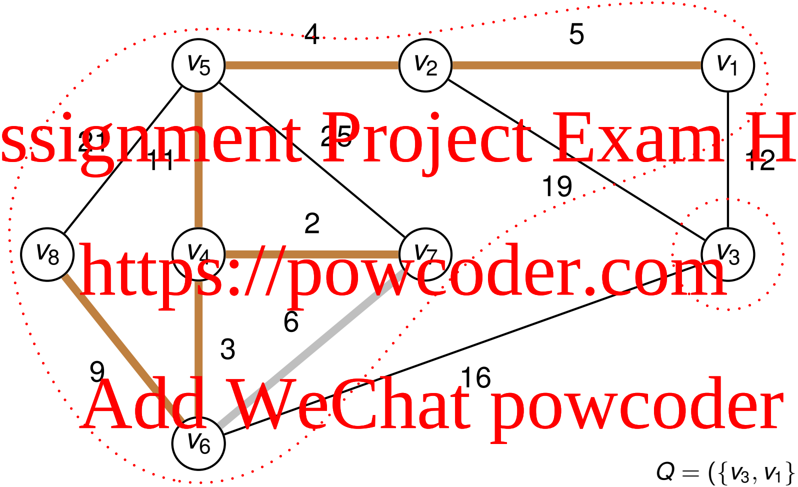
Illustration of Kruskal's Algorithm



$Q = (\{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

$C(v_4) \neq C(v_5)$ so add to E' and combine clusters

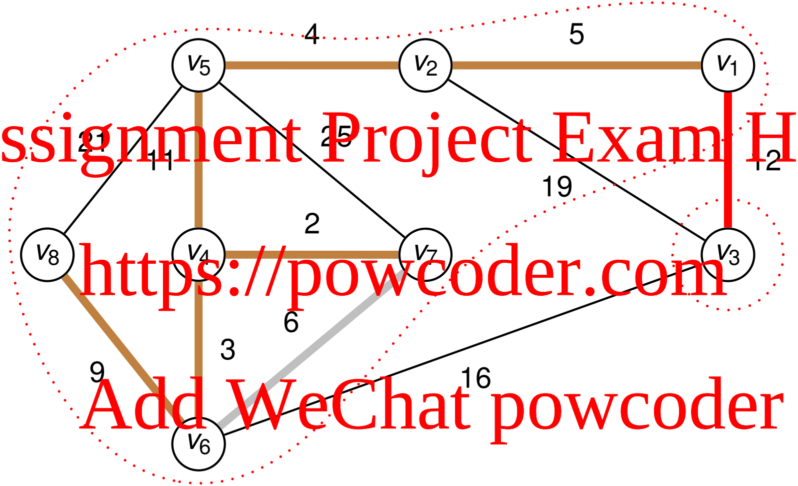
Illustration of Kruskal's Algorithm



$Q = (\{v_3, v_1\},$
$\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

Remove edge $\{v_3, v_1\}$ from Q

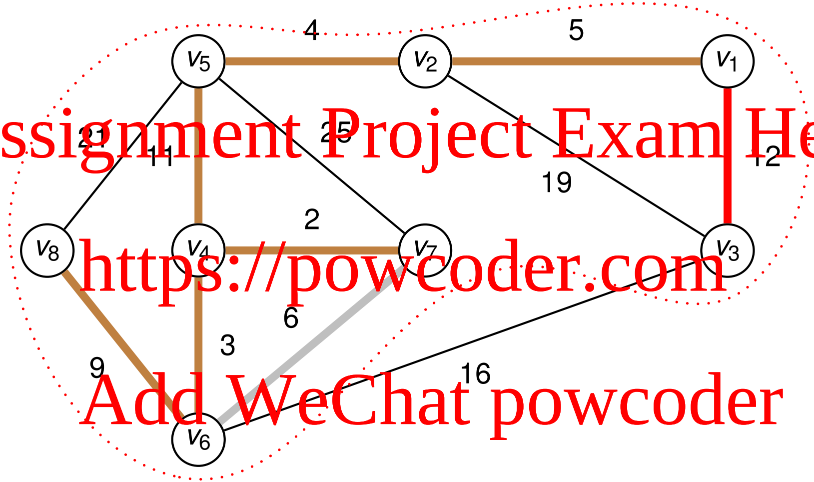
Illustration of Kruskal's Algorithm



$Q = (\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

$C(v_1) \neq C(v_3)$ so add to E' and combine clusters

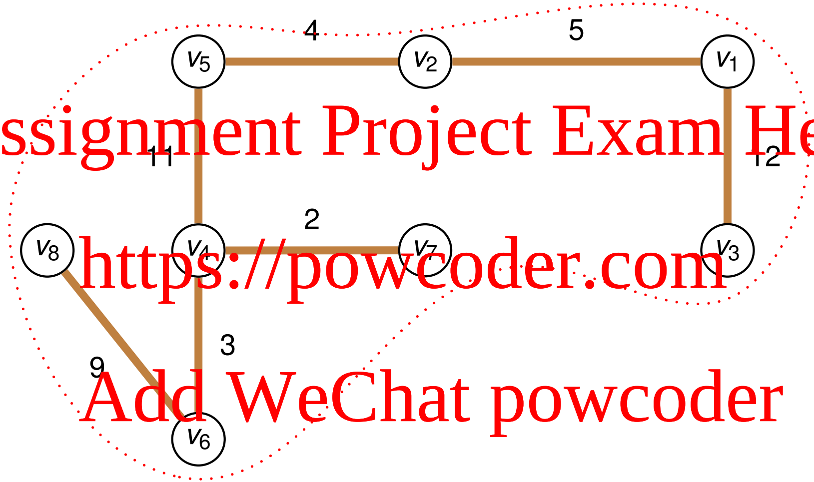
Illustration of Kruskal's Algorithm



$Q = (\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$

$|E'| = |V| - 1$ so we have found our minimum spanning tree

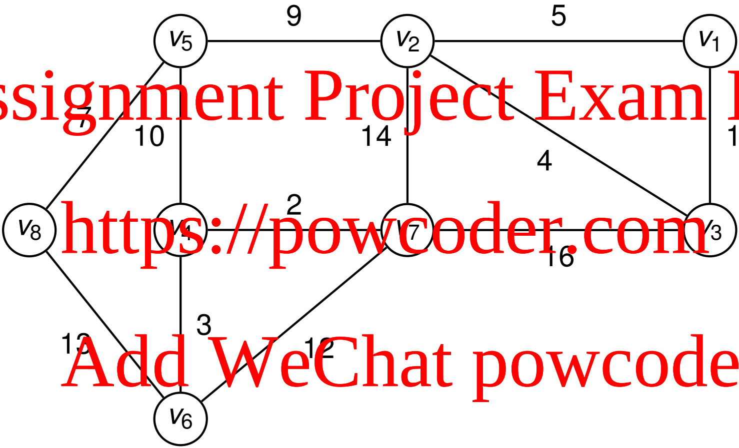
Illustration of Kruskal's Algorithm



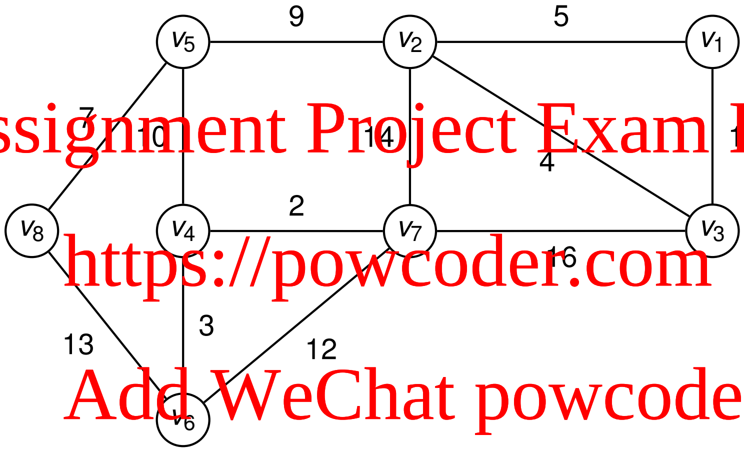
$$Q = (\{v_3, v_6\}, \{v_3, v_2\}, \{v_5, v_8\}, \{v_5, v_7\})$$

All done!

An example for you



An example for you



MST contains following edges (in order added):
$\{v_1, v_3\}, \{v_4, v_7\}, \{v_4, v_6\}, \{v_2, v_3\}, \{v_5, v_8\}, \{v_5, v_2\}, \{v_5, v_4\}$

Proof of Correctness of Kruskal's Algorithm

Follows from the Cut Property

- Suppose that edge $\{u, v\}$ is added by the algorithm
- Let V_1 be the vertices in $C(u)$ and V_2 be all the other vertices
- Prior to adding $\{u, v\}$ to E' there are no paths involving edges in E' from vertices in V_1 to vertices in V_2
- Since edges are being considered in increasing order of weight, $\{u, v\}$ must be the edge with the least weight connecting a vertex in V_1 with one in V_2
- Hence, by the Cut Property, $\{u, v\}$ is in every minimum spanning tree of the graph

Kruskal's MST Algorithm

Kruskal(G, w) :
  let Q *be the edges in* E *ordered by increasing weight*
  initialise E' *to be the empty set*
  for each vertex $v \in V$
    let $C(v)$ *be the singleton set containing* v
  while $|E'| < |V| - 1$
    remove edge $\{u, v\}$ *from* Q
    if $C(u) \neq C(v)$ *then*
      include $\{u, v\}$ *in* E'
      $C(u) \leftarrow C(v) \leftarrow C(u) \cup C(v)$
  return E'

Running Time of Kruskal's Algorithm

Measure of progress:

- Measure of progress is the number of edges left in Q
- Reduces by 1 every time through the loop
- Gives upper bound on number of iterations of $O(m)$

Running Time of Kruskal's Algorithm (cont.)

Initialising Q:

- Creating Q involves sorting E
- E contains m elements
- Sorting m elements can be done in $O(m \log m)$ steps
- Note that m is $O(n^2)$
- Furthermore, $\log n^2 = 2 \log n$
- Hence $O(\log m) = O(\log n)$
- Hence the number of steps to create Q is $O(m \log n)$

Running Time of Kruskal's Algorithm (cont.)

Optimising the representation of clusters (sets)

Question: Which of set operations do we need to be particularly fast?

Answer:

- Checking equality of two sets
- Producing the union of two sets

This is the so-called **Union-Find** data structure

We will look at this data structure in an exercise class

We can achieve an overall running time for the algorithm of $O(m \log n)$

Another MST Algorithm

- Discovered by Jarník in 1930
- Rediscovered by Prim in 1957
- Rediscovered (again) by Dijkstra in 1959
- Most commonly known as Prim's algorithm!

Jarník's Algorithm

- Grows the tree E' from an arbitrary starting point
- As E' grows, it **spans** an increasing number of the vertices
- E' grows by one at each iteration
- Selects the edge that connects the **closest** vertex to the tree produced so far
- Committing to this edge is *safe* due to the Cut Property

The Closest Vertex

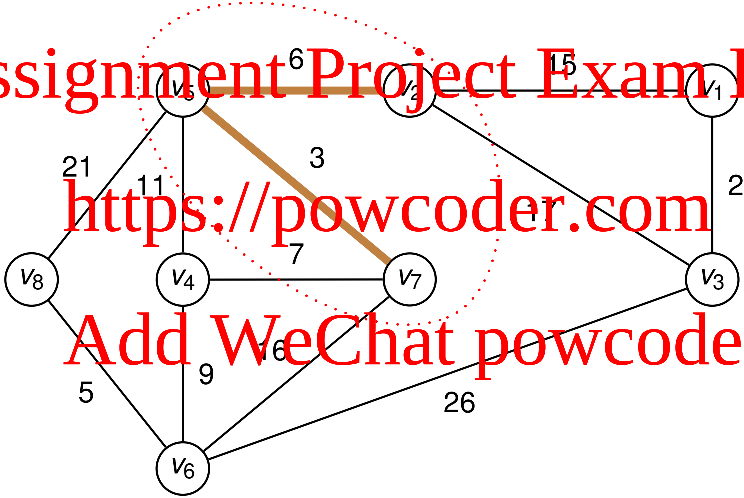
- Consider a graph $G = (V, E)$
- Suppose the algorithm has so far selected the set of edges E'
- E' spans the vertices S where $S \subseteq V$
- In other words $G' = (S, E')$ is a spanning tree

Question: Which is the closest vertex to (S, E')?

Answer: Vertex v which minimises $w(u, v)$ where $u \in S$ and $v \notin S$

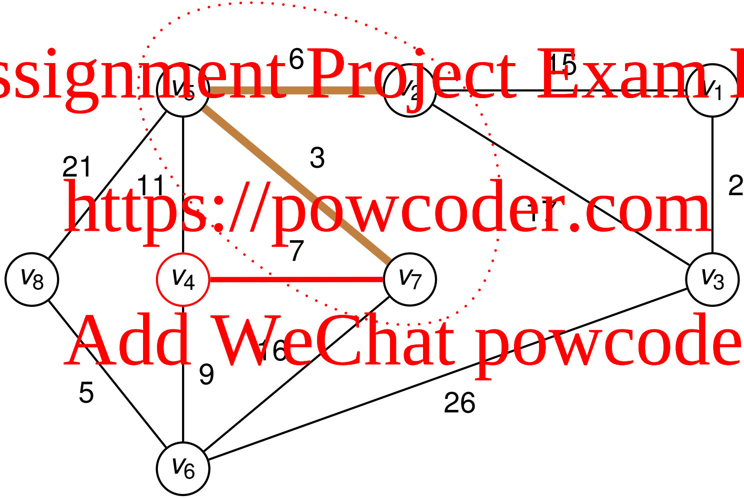
Example

In this example we have $E' = \{\{v_5, v_7\}, \{v_5, v_2\}\}$ and $S = \{v_2, v_5, v_7\}$



Example

The closest vertex to (S, E') is v_4 due to the edge $\{v_4, v_7\}$



Maintaining Distance from (S, E')

- Record distance of v from (S, E') as value

$$\delta(v)$$

- As E' grows $\delta(v)$ may reduce
- Also useful to record identify of the vertex $u \in S$ where

$$\delta(v) = w(v, u)$$

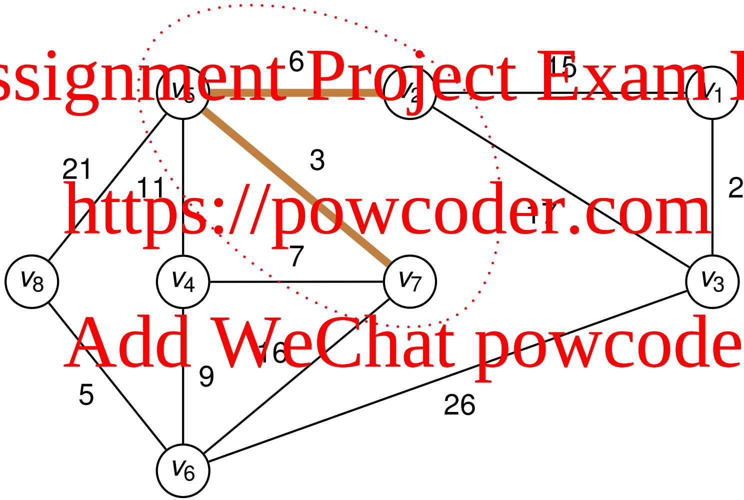
refer to this value as

$$\beta(v)$$

- Back to the example

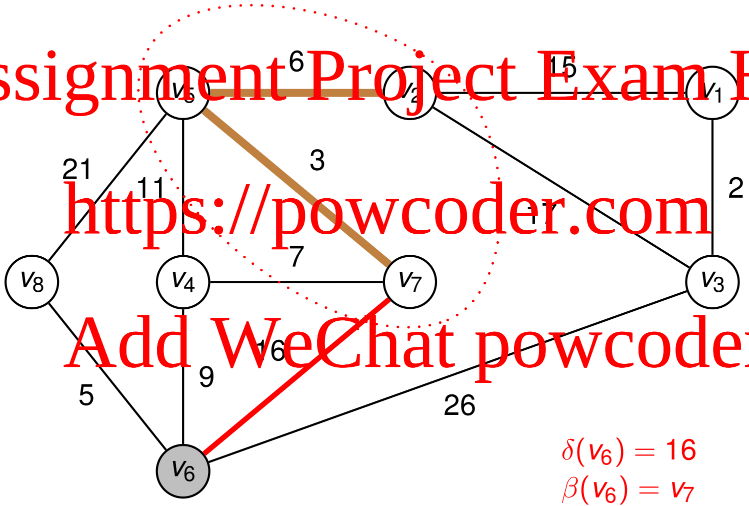
Example

Recall that we have $E' = \{\{v_5, v_7\}, \{v_5, v_2\}\}$ and $S = \{v_2, v_5, v_7\}$



Example

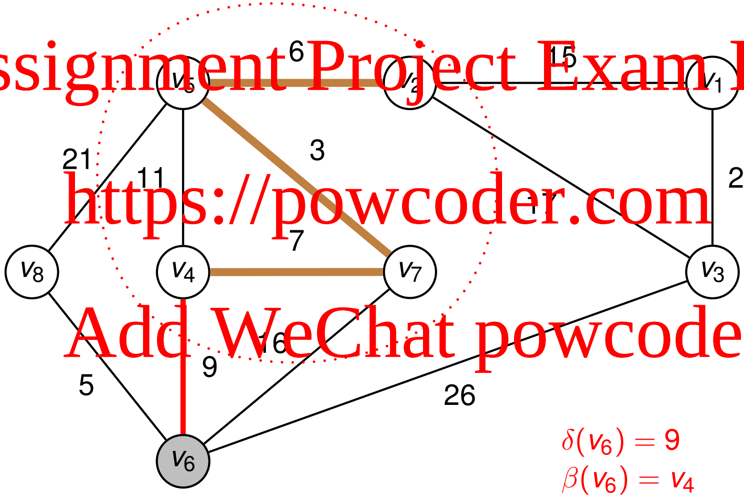
Consider the values $\delta(v_6)$ and $\beta(v_6)$ given this particular (T, S)



$\delta(v_6) = 16$
$\beta(v_6) = v_7$

Example

What happens to $\delta(v_6)$ when we were to add v_4 to E'



$\delta(v_6) = 9$
$\beta(v_6) = v_4$

Updating Distances

In order to maintain the values of δ as the size of E' grows the algorithm must do the following:

> *When adding u into S (by adding the edge involving u into E') must check all edges from u to some vertices v not yet in S to see if $\delta(v)$ can be reduced*

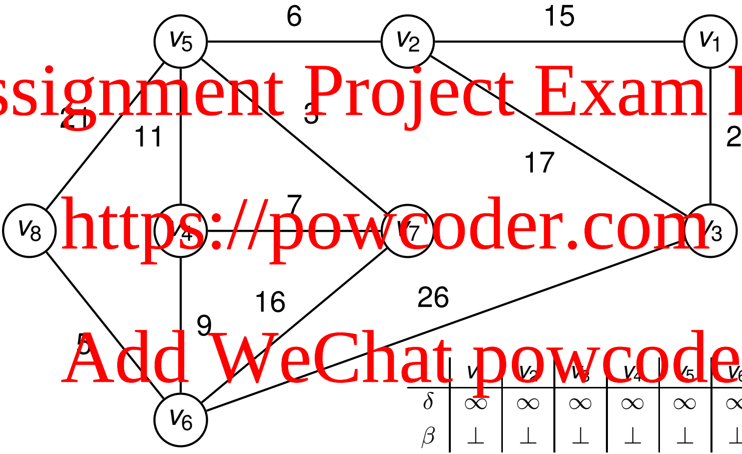
As in Dijkstra's Algorithm, we maintain a priority queue of vertices with lower δ values giving higher priority

Jarník's Algorithm

Jarník(G, w) :

select some vertex $s \in V$ *and let* $\delta(s) = 0$
initialise E' *to be the set* $\{\}$
for all $v \in V - \{s\}$ *let* $\delta(v) = \infty$
for all $v \in V$ *let* $\beta(v) = \perp$
let Q *be a priority queue containing elements of* V
while $|E'| < |V| - 1$
    remove u *from front of priority queue* Q
    if $\beta(u) \neq \perp$ *then*   *(% i.e.* u *is not* s*)*
        add $\{u, v\}$ *to* E' *where* $\beta(u) = v$
    for each $\{u, x\} \in E$ *where* x *remains in* Q
        if $\delta(x) > w(u, x)$ *then*
            let $\delta(x) = w(u, x)$
            let $\beta(x) = u$
return E'

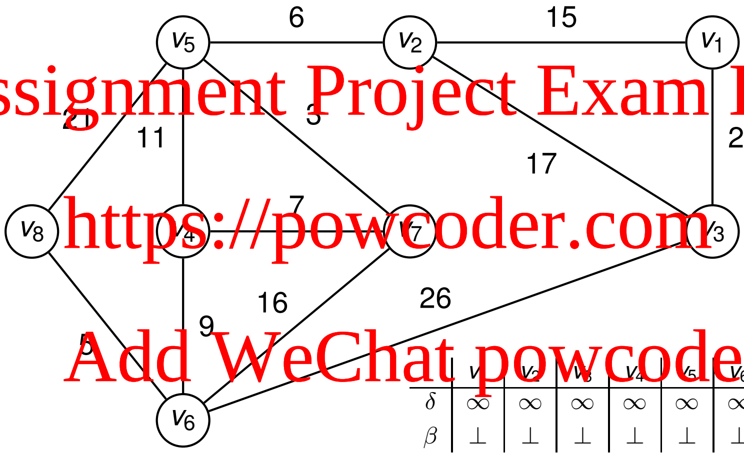
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	∞	∞	∞	0	∞
β	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Let $s = v_7$, $E' = \{\}$, and initialise δ and β as shown

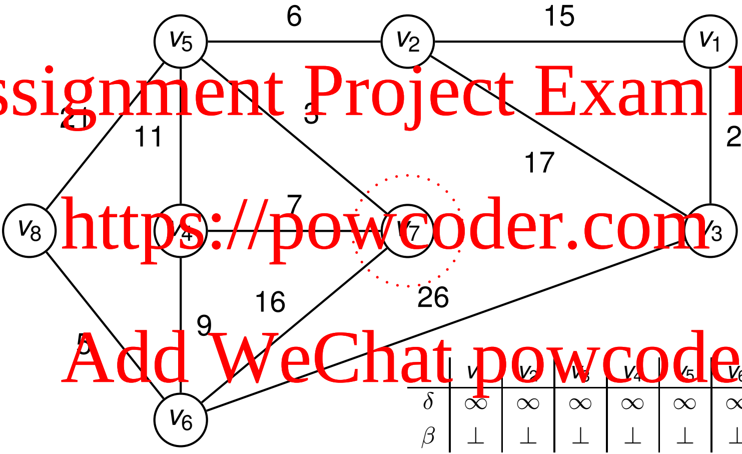
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	∞	∞	∞	0	∞
β	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Remove v_7 from Q

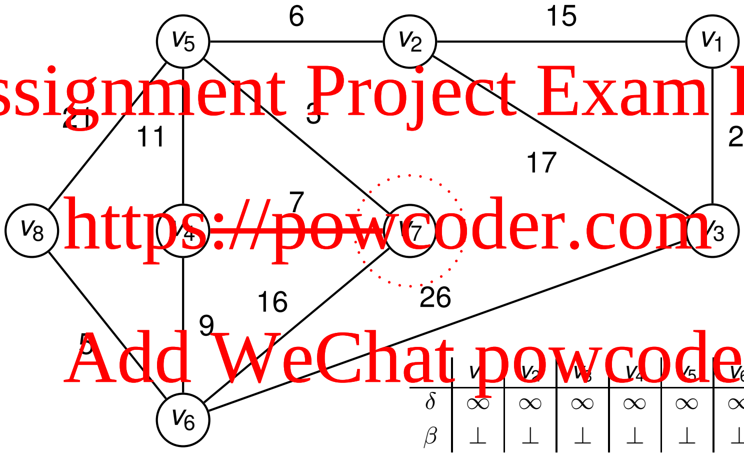
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	∞	∞	∞	0	∞
β	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Consider the edge $\{v_7, v_4\}$

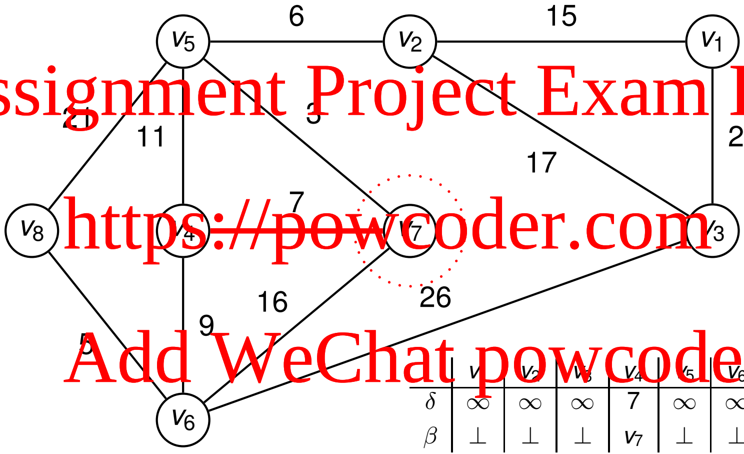
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	∞	∞	∞	0	∞
β	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

$\delta(v_4) > w(v_7, v_4)$ so update $\delta(v_4)$ and $\beta(v_4)$

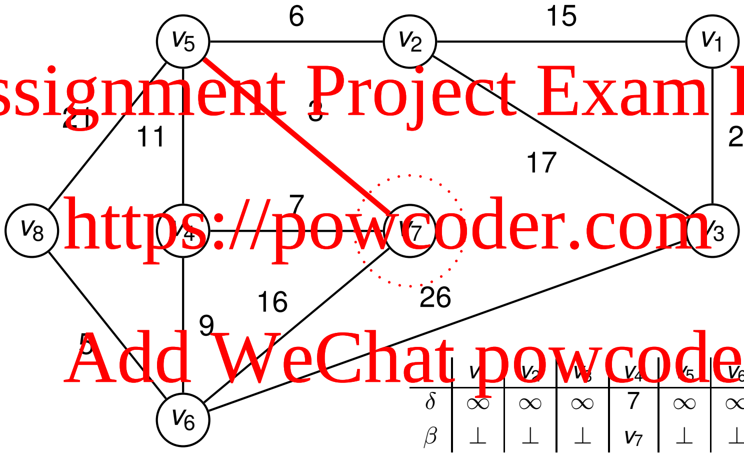
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	7	∞	∞	0	∞
β	$\perp$	$\perp$	$\perp$	v_7	$\perp$	$\perp$	$\perp$	$\perp$

Consider the edge $\{v_7, v_5\}$

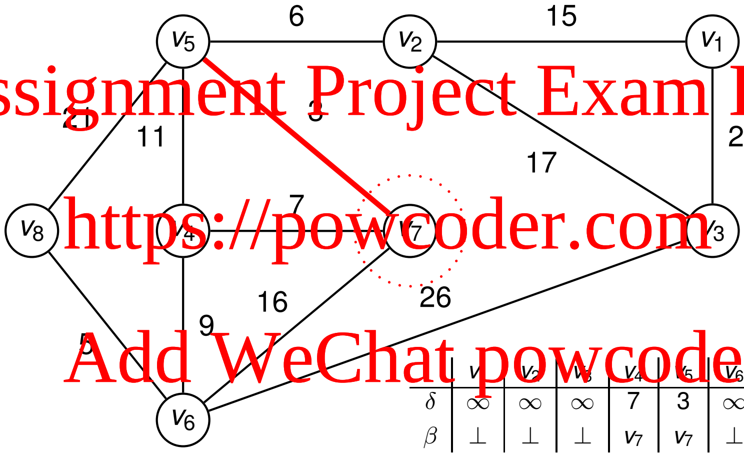
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	7	∞	∞	0	∞
β	$\perp$	$\perp$	$\perp$	v_7	$\perp$	$\perp$	$\perp$	$\perp$

$\delta(v_5) > w(v_7, v_5)$ so update $\delta(v_5)$ and $\beta(v_5)$

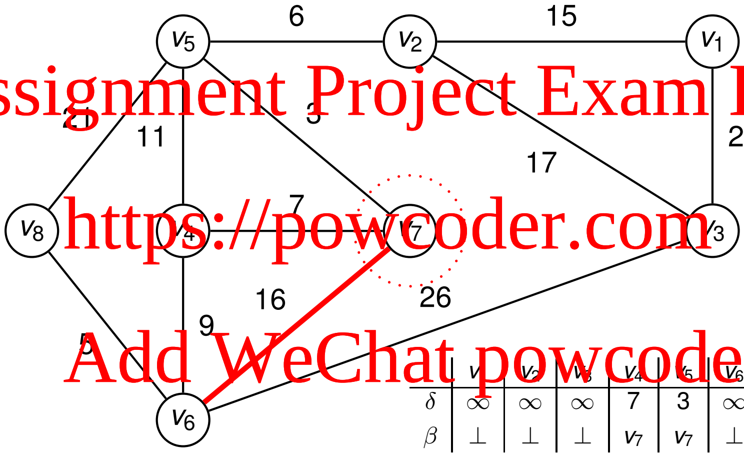
Illustrating Jarník’s Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	7	3	∞	0	∞
β	$\perp$	$\perp$	$\perp$	v_7	v_7	$\perp$	$\perp$	$\perp$

Consider the edge $\{v_7, v_6\}$

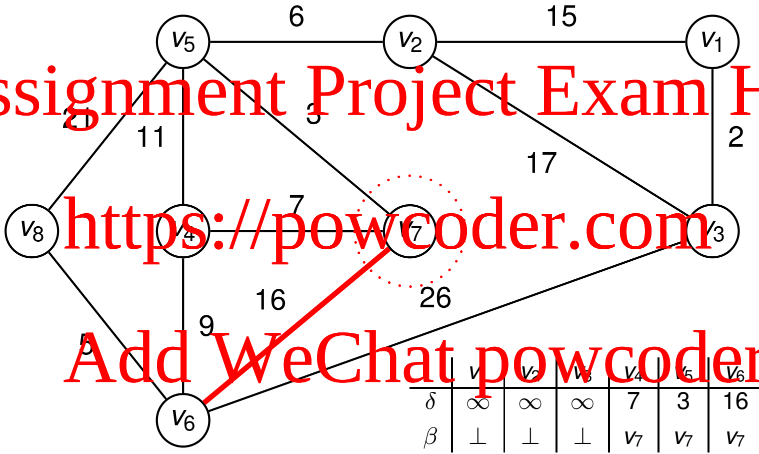
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	7	3	∞	0	∞
β	$\perp$	$\perp$	$\perp$	v_7	v_7	$\perp$	$\perp$	$\perp$

$\delta(v_6) > w(v_7, v_6)$ so update $\delta(v_6)$ and $\beta(v_6)$

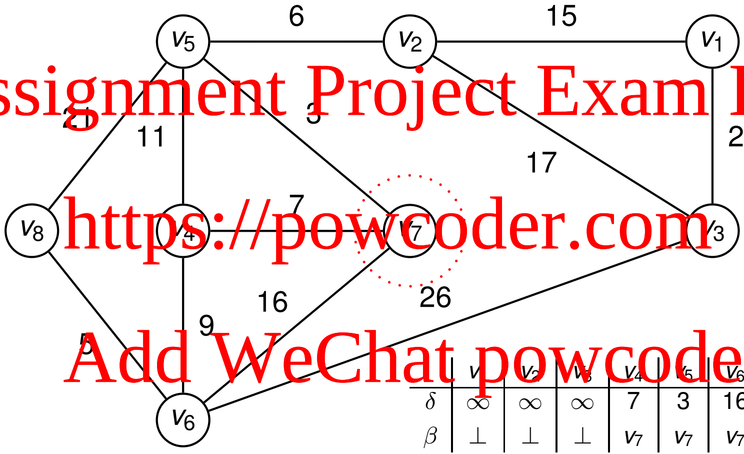
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	7	3	16	0	∞
β	$\perp$	$\perp$	$\perp$	v_7	v_7	v_7	$\perp$	$\perp$

All edges from v_7 considered

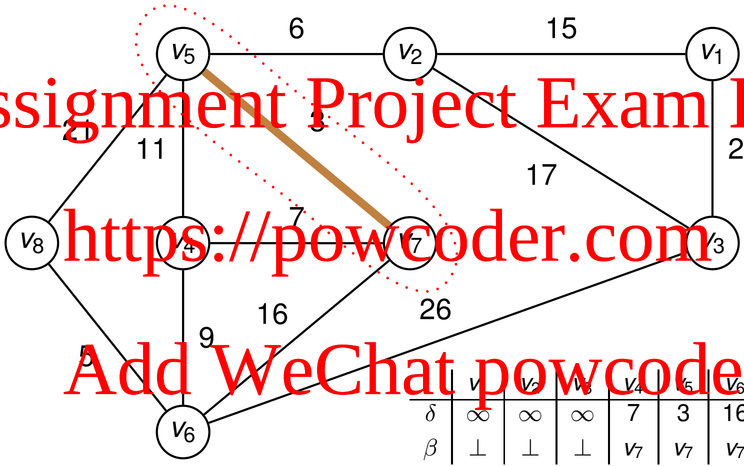
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	7	3	16	0	∞
β	$\perp$	$\perp$	$\perp$	v_7	v_7	v_7	$\perp$	$\perp$

Remove v_5 from Q and add $\{v_5, v_7\}$ to E'

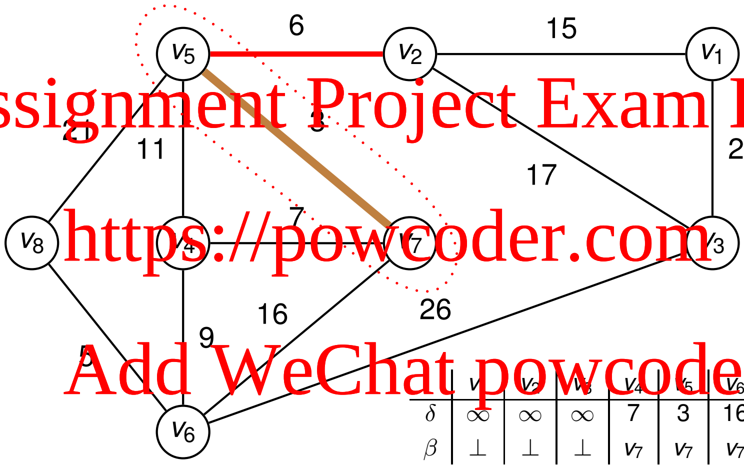
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	7	3	16	0	∞
β	$\perp$	$\perp$	$\perp$	v_7	v_7	v_7	$\perp$	$\perp$

Consider the edge $\{v_5, v_2\}$

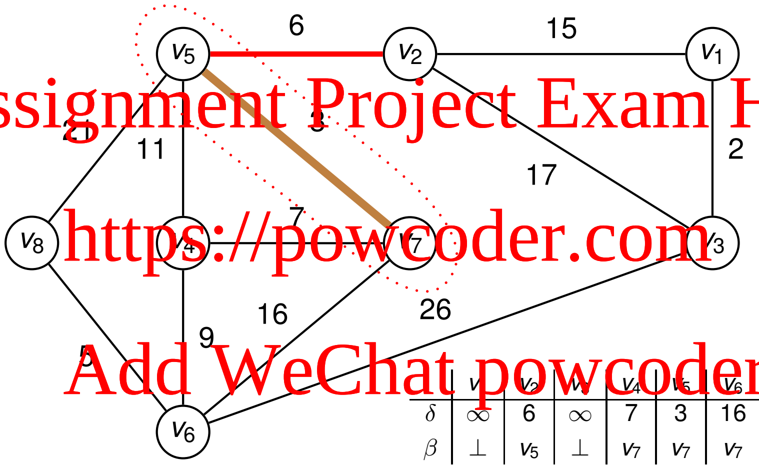
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	∞	∞	7	3	16	0	∞
β	$\perp$	$\perp$	$\perp$	v_7	v_7	v_7	$\perp$	$\perp$

$\delta(v_2) > w(v_5, v_2)$ so update $\delta(v_2)$ and $\beta(v_2)$

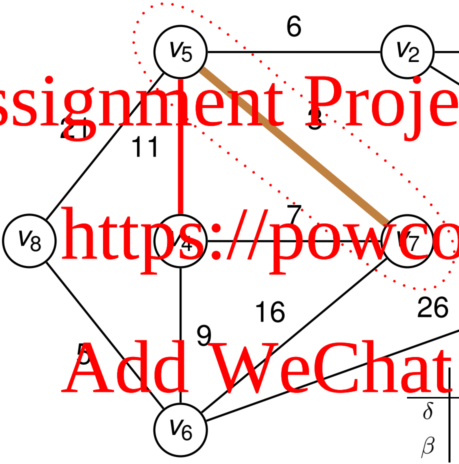
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	6	∞	7	3	16	0	∞
β	$\perp$	v_5	$\perp$	v_7	v_7	v_7	$\perp$	$\perp$

Consider the edge $\{v_5, v_4\}$

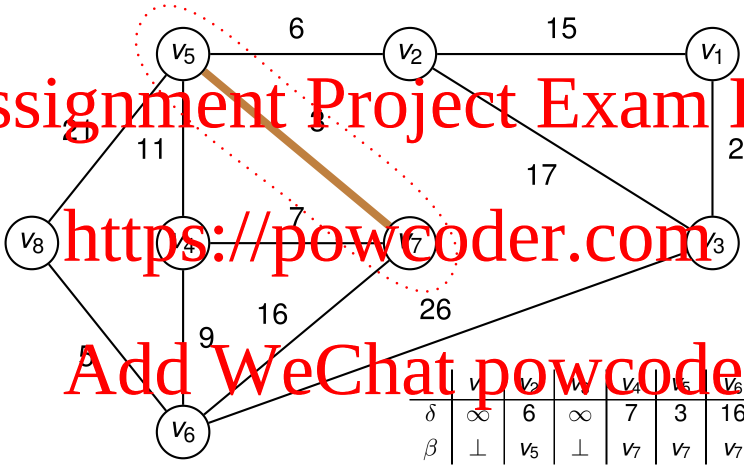
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	6	∞	7	3	16	0	∞
β	$\perp$	v_5	$\perp$	v_7	v_7	v_7	$\perp$	$\perp$

$\delta(v_4) < w(v_4, v_5)$ so don't update $\delta(v_4)$ or $\beta(v_4)$

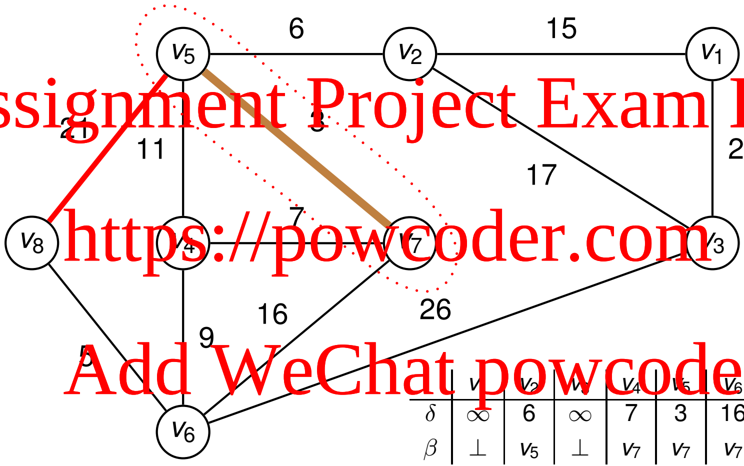
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	6	∞	7	3	16	0	∞
β	$\perp$	v_5	$\perp$	v_7	v_7	v_7	$\perp$	$\perp$

Consider the edge $\{v_5, v_8\}$

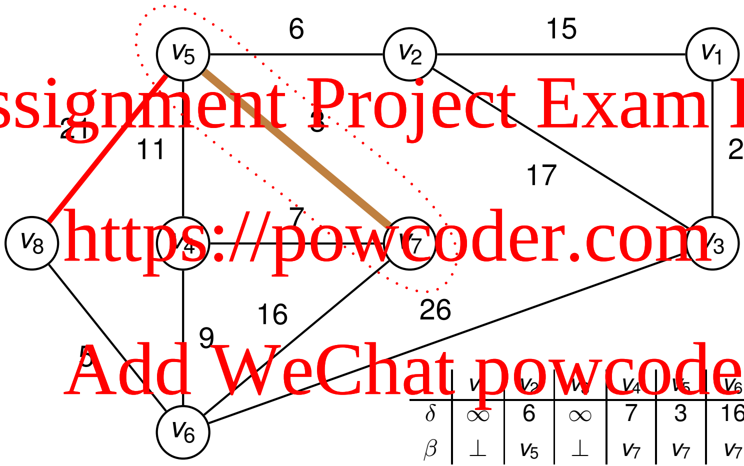
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	6	∞	7	3	16	0	∞
β	$\perp$	v_5	$\perp$	v_7	v_7	v_7	$\perp$	$\perp$

$\delta(v_8) > w(v_5, v_8)$ so update $\delta(v_8)$ and $\beta(v_8)$

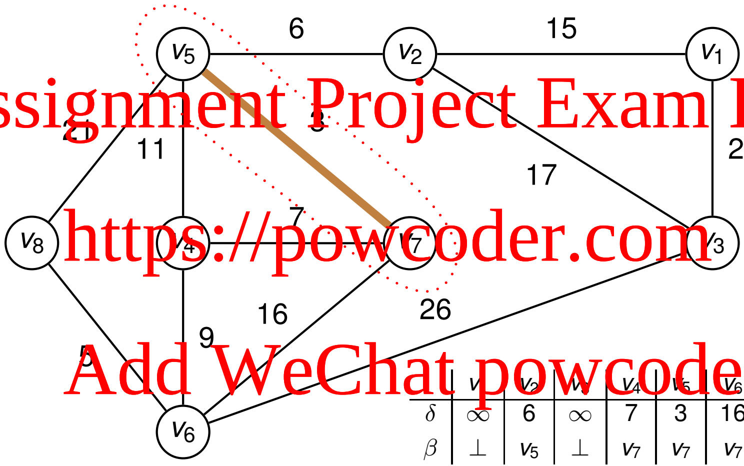
Illustrating Jarník’s Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	6	∞	7	3	16	0	21
β	$\perp$	v_5	$\perp$	v_7	v_7	v_7	$\perp$	v_5

All edges from v_5 considered

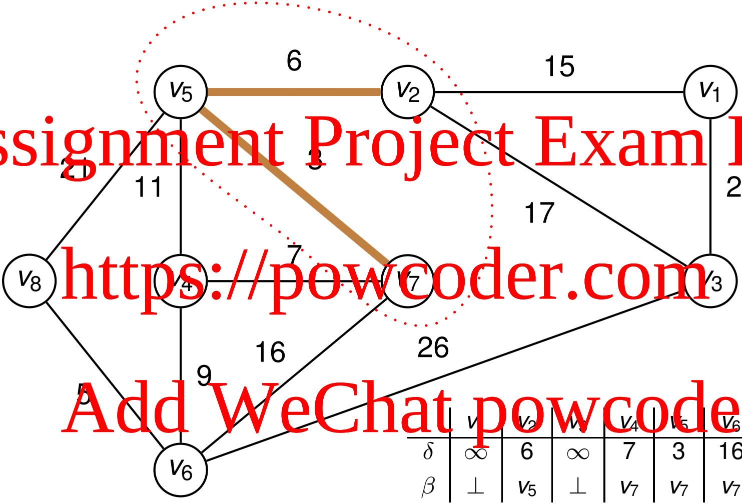
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	6	∞	7	3	16	0	21
β	$\perp$	v_5	$\perp$	v_7	v_7	v_7	$\perp$	v_5

Remove v_2 from Q and add edge $\{v_2, v_5\}$ to E'

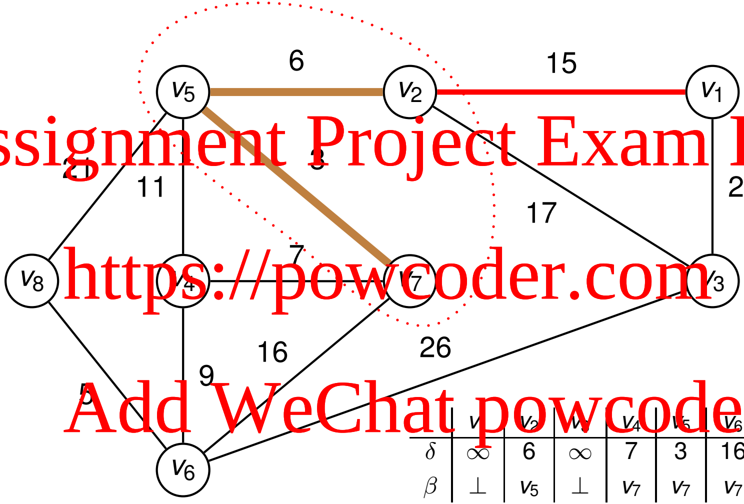
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	6	∞	7	3	16	0	21
β	$\perp$	v_5	$\perp$	v_7	v_7	v_7	$\perp$	v_5

Consider edge $\{v_2, v_1\}$

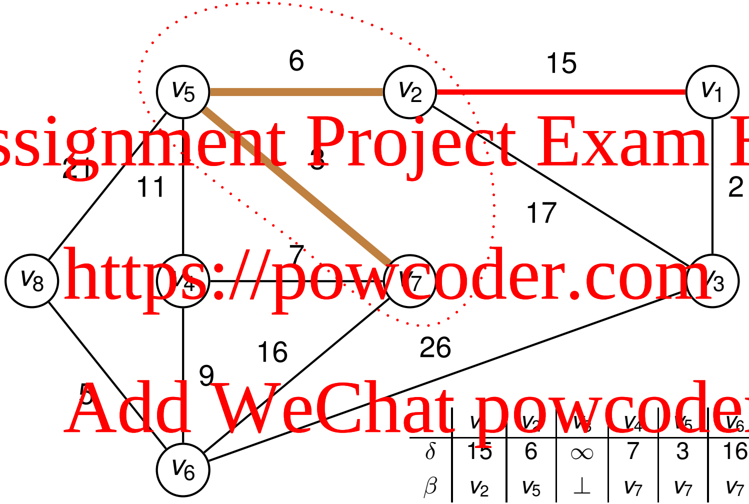
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	∞	6	∞	7	3	16	0	21
β	$\perp$	v_5	$\perp$	v_7	v_7	v_7	$\perp$	v_5

$\delta(v_1) > w(v_2, v_1)$ so update $\delta(v_1)$ and $\beta(v_1)$

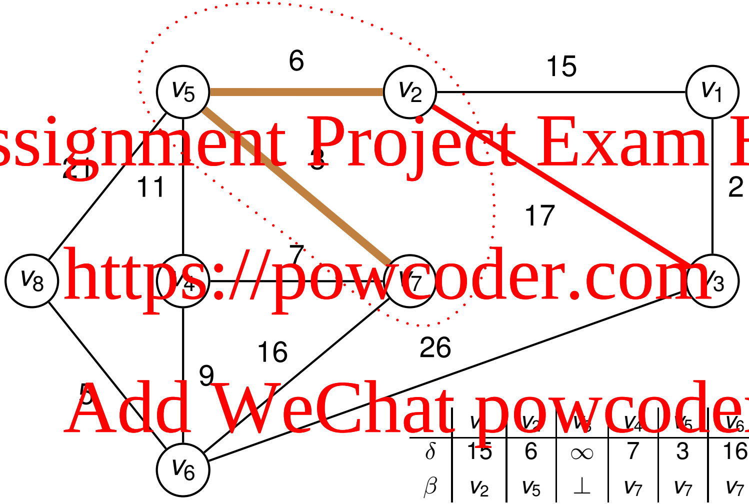
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	∞	7	3	16	0	21
β	v_2	v_5	$\perp$	v_7	v_7	v_7	$\perp$	v_5

Consider edge $\{v_2, v_3\}$

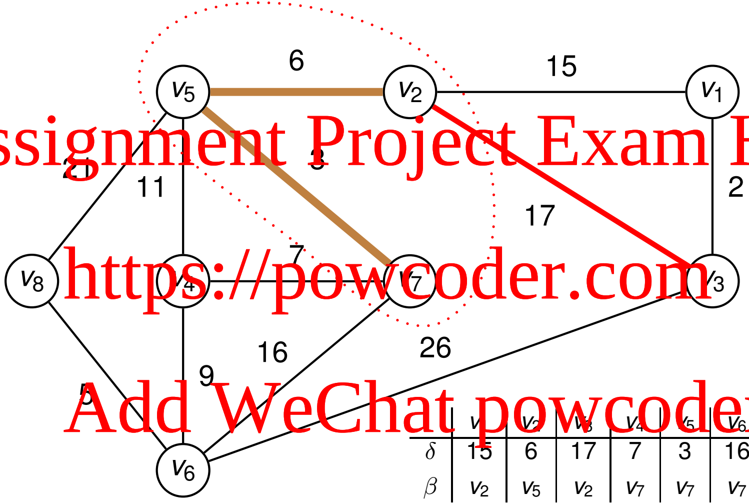
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	∞	7	3	16	0	21
β	v_2	v_5	$\perp$	v_7	v_7	v_7	$\perp$	v_5

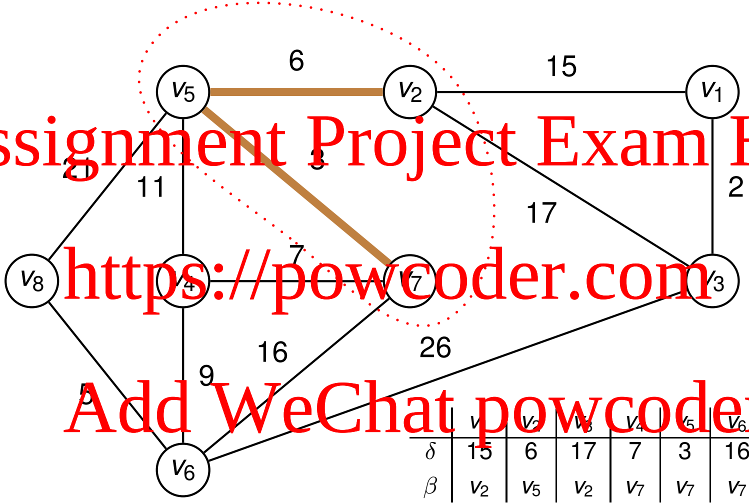
$\delta(v_3) > w(v_2, v_3)$ so update $\delta(v_3)$ and $\beta(v_3)$

Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	16	0	21
β	v_2	v_5	v_2	v_7	v_7	v_7	$\perp$	v_5

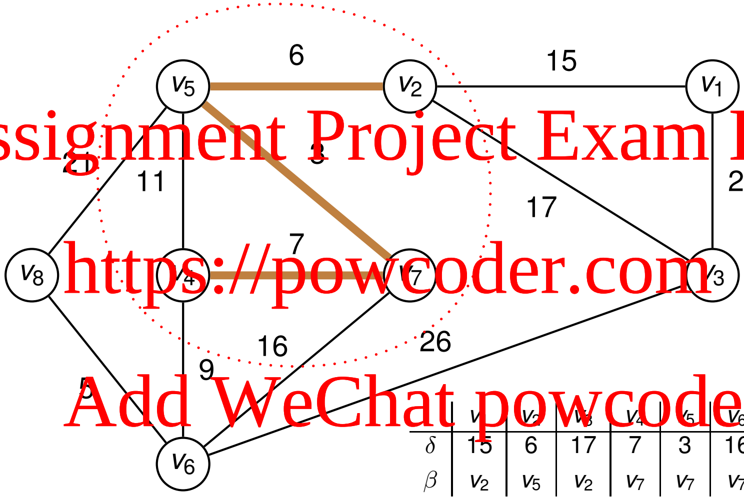
All edges from v_2 considered



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	16	0	21
β	v_2	v_5	v_2	v_7	v_7	v_7	$\perp$	v_5

Remove v_4 from Q and add edge $\{v_4, v_7\}$ to E'

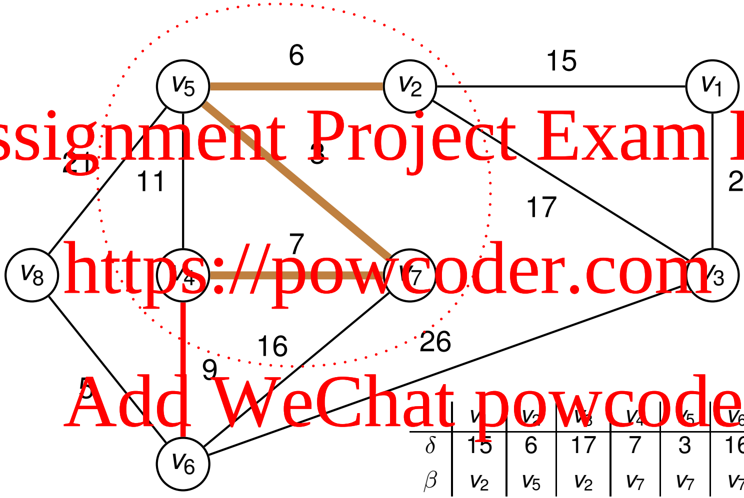
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	16	0	21
β	v_2	v_5	v_2	v_7	v_7	v_7	$\perp$	v_5

Consider edge $\{v_4, v_6\}$

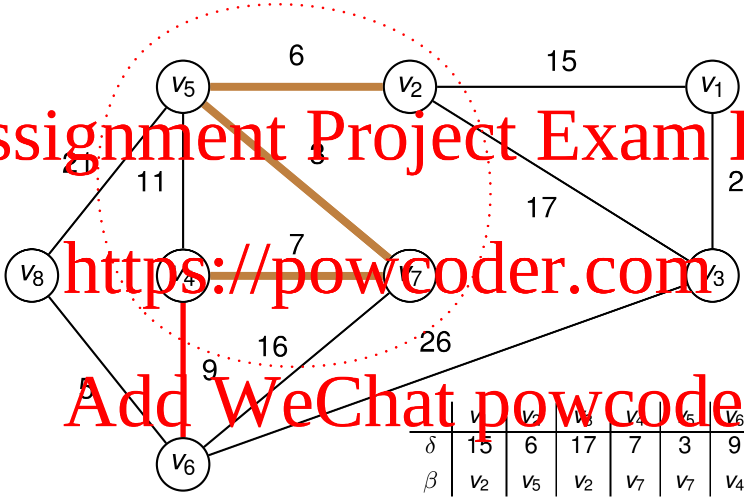
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	16	0	21
β	v_2	v_5	v_2	v_7	v_7	v_7	$\perp$	v_5

$\delta(v_6) > w(v_4, v_6)$ so update $\delta(v_6)$ and $\beta(v_6)$

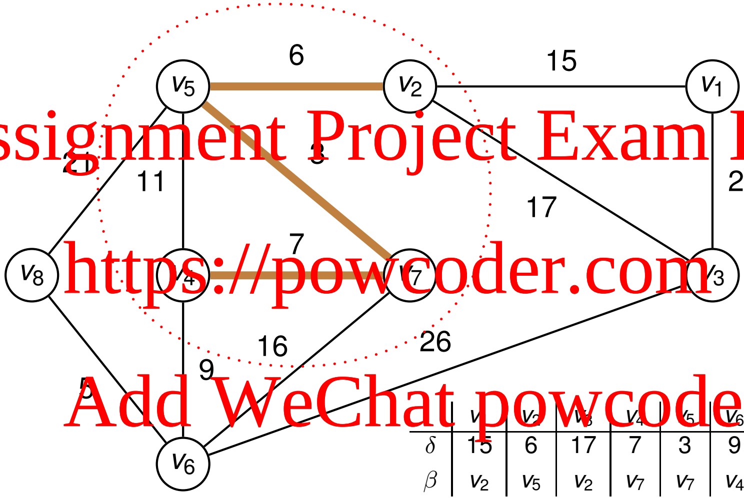
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	21
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_5

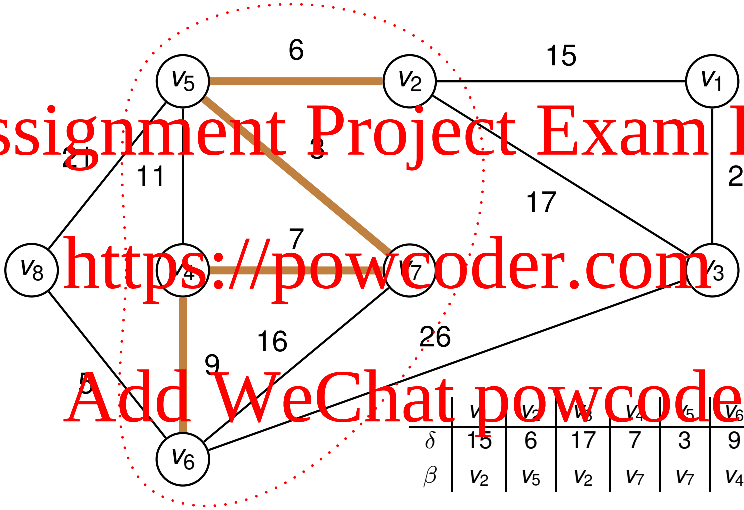
All edges from v_4 considered

Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	21
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_5

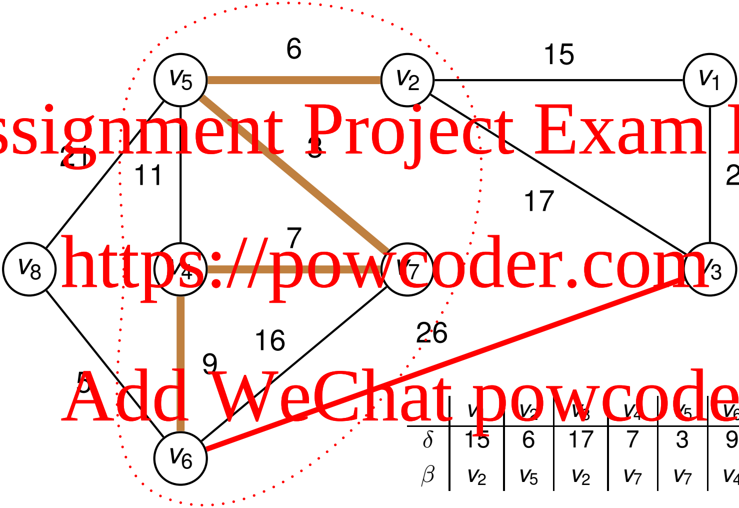
Remove v_6 from Q and add edge $\{v_6, v_4\}$ to E'



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	21
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_5

Consider edge $\{v_6, v_3\}$

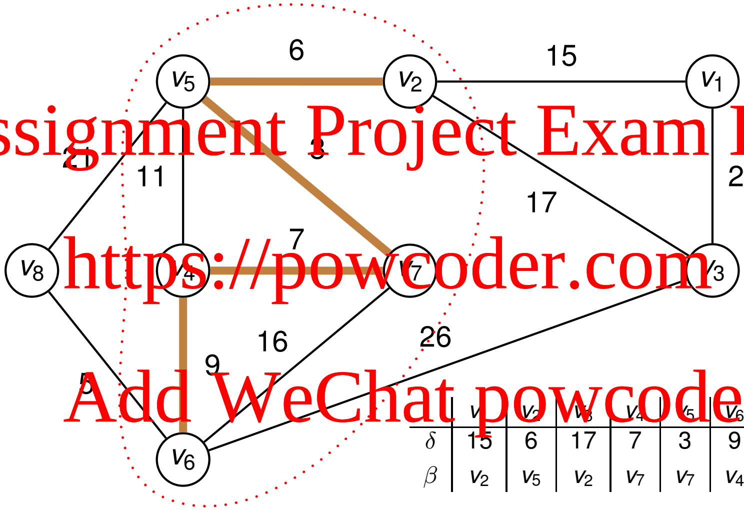
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	21
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_5

$\delta(v_3) < w(v_6, v_3)$ so don't update $\delta(v_3)$ and $\beta(v_3)$

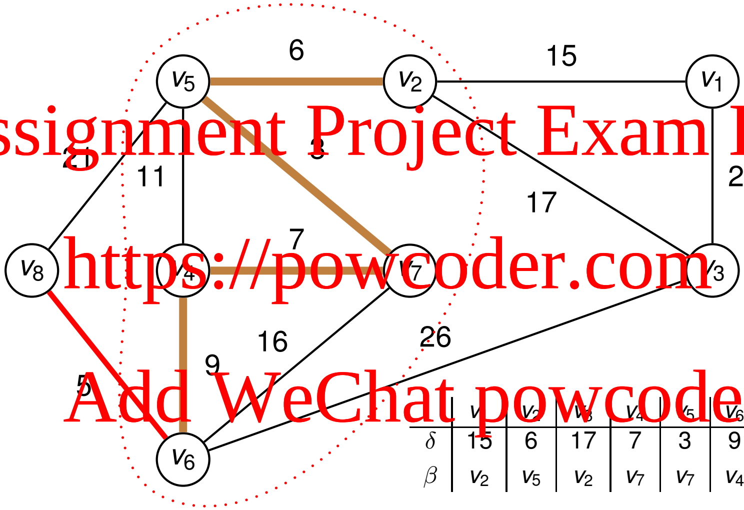
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	21
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_5

Consider edge $\{v_6, v_8\}$

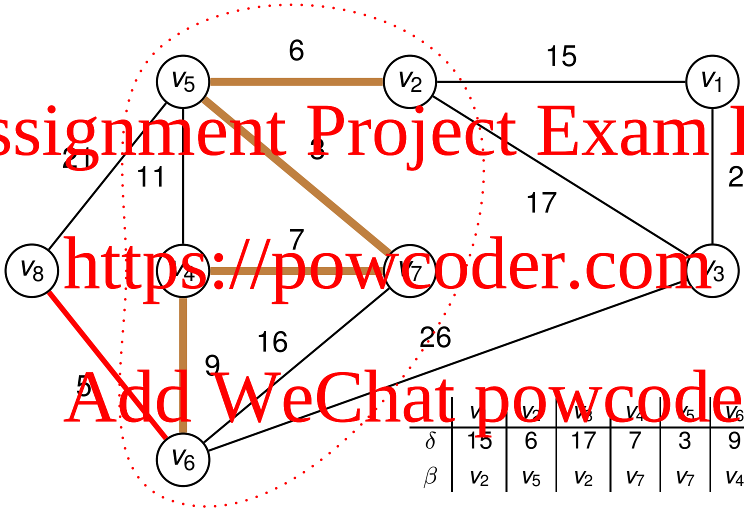
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	21
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_5

$\delta(v_8) > w(v_6, v_8)$ so update $\delta(v_8)$ and $\beta(v_8)$

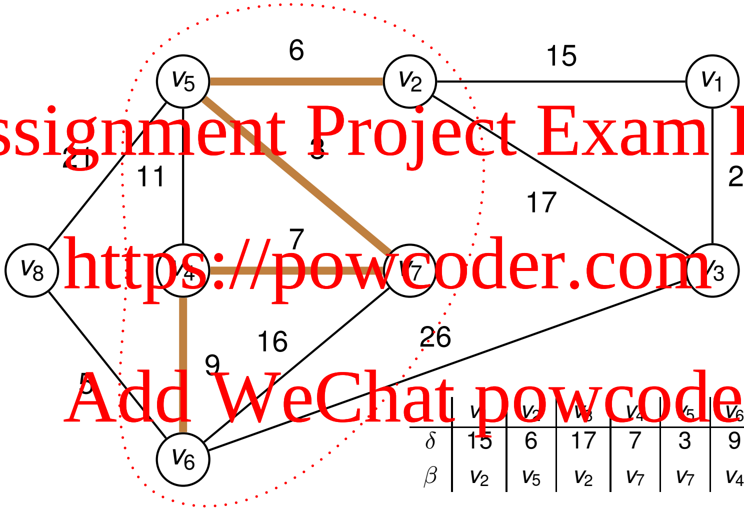
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	5
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_6

All edges from v_6 considered

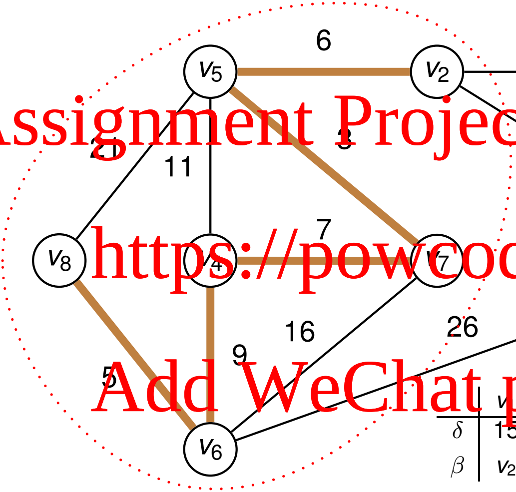
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	5
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_6

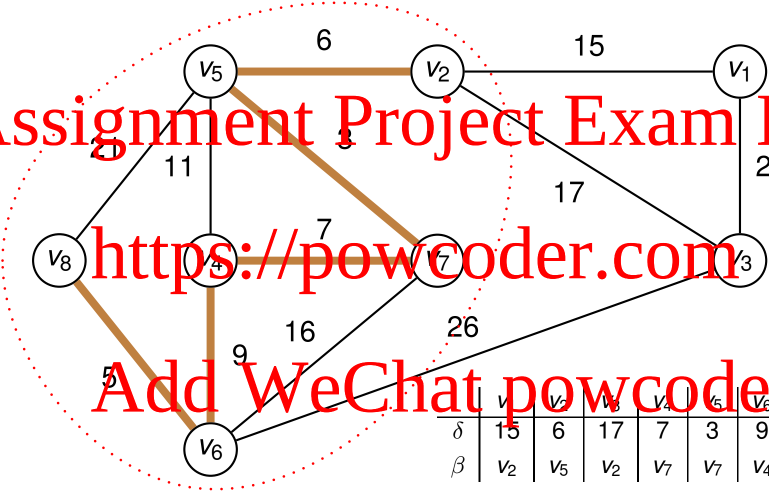
Remove v_8 from Q and add edge $\{v_8, v_6\}$ to E'

Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	5
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_6

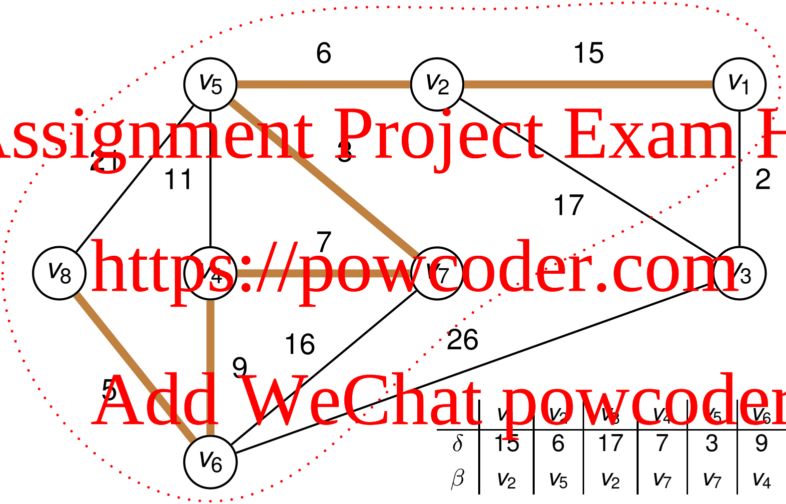
No edges from v_8 need to be considered



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	5
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_6

Remove v_1 from Q and add edge $\{v_1, v_2\}$ to E'

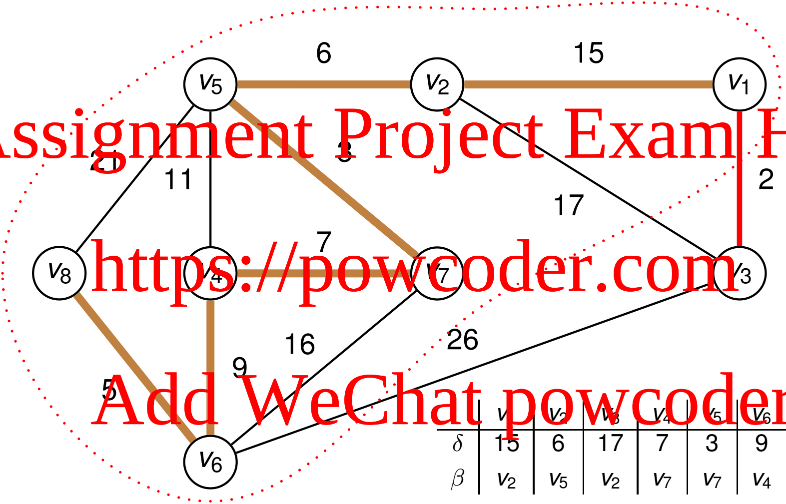
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	5
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_6

Consider edge $\{v_1, v_3\}$

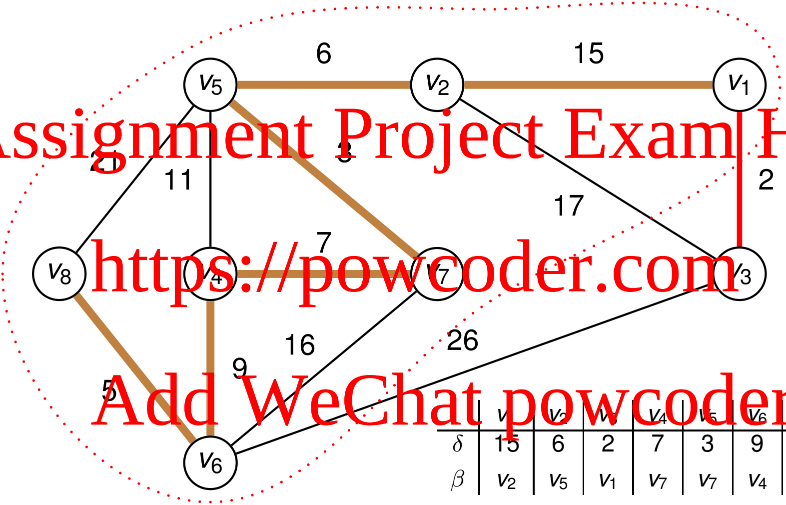
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	17	7	3	9	0	5
β	v_2	v_5	v_2	v_7	v_7	v_4	$\perp$	v_6

$\delta(v_3) > w(v_1, v_3)$ so update $\delta(v_3)$ and $\beta(v_3)$

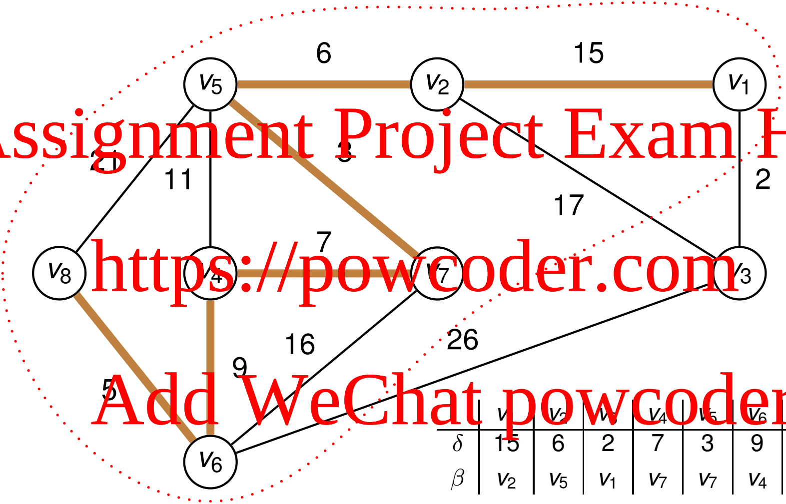
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	2	7	3	9	0	5
β	v_2	v_5	v_1	v_7	v_7	v_4	$\perp$	v_6

All edges from v_1 considered

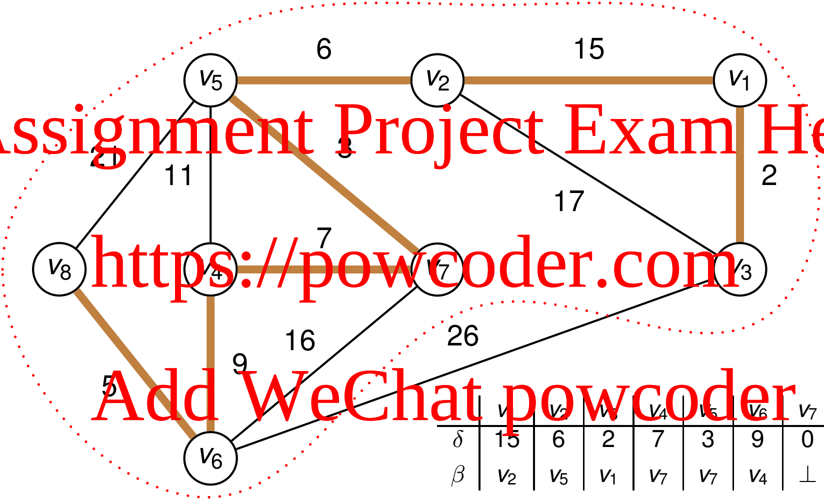
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	2	7	3	9	0	5
β	v_2	v_5	v_1	v_7	v_7	v_4	$\perp$	v_6

Remove v_3 from Q and add edge $\{v_3, v_1\}$ to E'

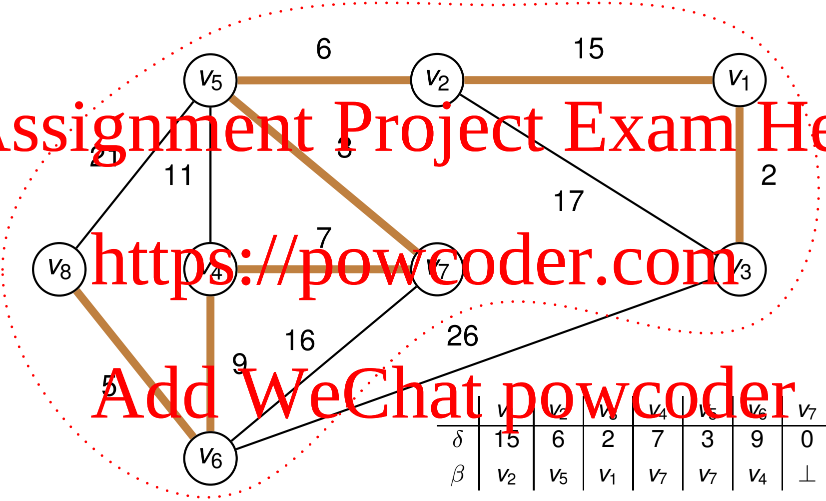
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	2	7	3	9	0	5
β	v_2	v_5	v_1	v_7	v_7	v_4	$\perp$	v_6

No edges from v_3 need to be considered

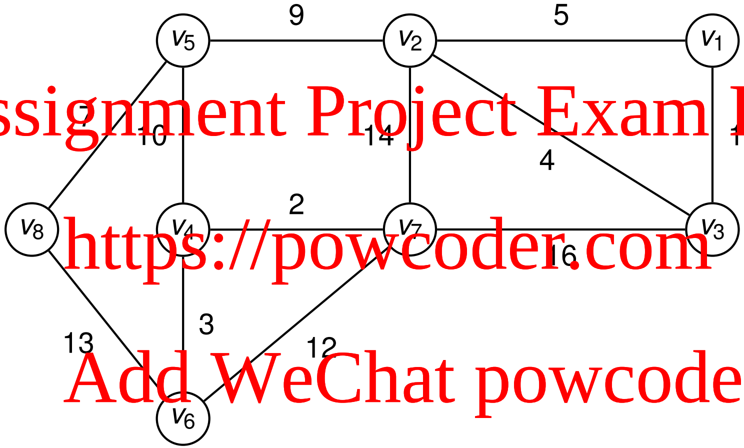
Illustrating Jarník's Algorithm



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
δ	15	6	2	7	3	9	0	5
β	v_2	v_5	v_1	v_7	v_7	v_4	$\perp$	v_6

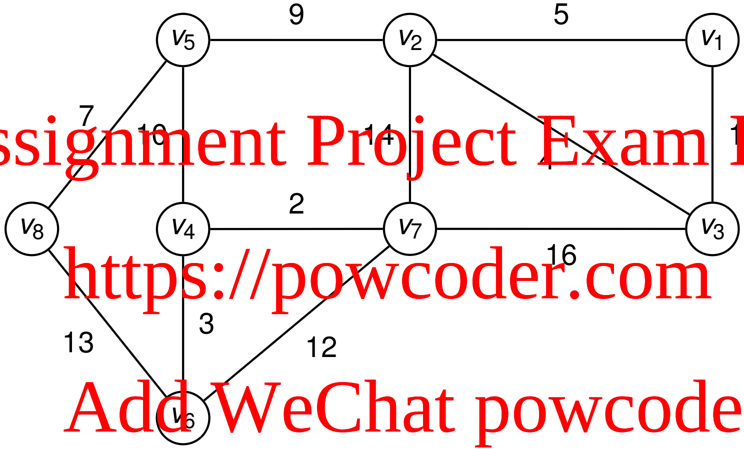
All done!

An example for you



Start with v_7

An example for you



Start with v_7
MST contains following edges (in order added):
$\{v_4, v_7\}, \{v_4, v_6\}, \{v_5, v_4\}, \{v_5, v_8\}$,‘ $\{v_5, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}$

Correctness of Jarník's Algorithm

- Follows from the Cut Property
- Let V_1 be the vertices no longer in Q
- Let V_2 be the vertices still in Q
- Let u the vertex at the front of Q and $\beta(u) = v$
- $\{u, v\}$ is least weighted edge with one end in V_1 and other in V_2
- $\{u, v\}$ can safely be selected for inclusion in E'

Running time of Jarník's Algorithm

- Analysis is identical to that for Dijkstra's Algorithm
- $O(m)$ updates to the value of δ and β
- Assume Q implemented as a heap
- Each update of δ takes $\log n$ time
- Total running time is $O(m \log n)$

Greedy Algorithms:
Huffman Codes

Efficient Transmission of Messages

- Message sent using an alphabet of characters
- How efficiently can messages involving these characters be encoded in binary?
- Efficiency determined by how many bits are needed to encode messages on average
- Huffman Codes are used to find optimally efficient ways to solve this problem

A Straightforward Encoding

Suppose we are using an alphabet Σ containing 64 characters:

- 26 lower case letters and 26 upper case letters
- space, tab, newline
- full-stop, comma, exclamation mark, question mark and dash
- round brackets and square brackets

A Straightforward Encoding (cont.)

- Each character x in Σ can be encoded using a distinct 6 bits
- We refer the encoding of x as $\gamma(x)$

x	$\gamma(x)$
a	000000
b	000001
c	000010
d	000011
$\vdots$	$\vdots$

Message length

- Sending n characters requires $6n$ bits
- Exactly 6 bits per character

But this does not exploit differences in character frequencies

- ‘e’, ‘t’, ‘a’, ‘i’, ‘n’, ‘o’, ‘s’ are far more common than ‘z’, ‘j’ and ‘x’

Encoding should depend on how frequent a character is

Character Probabilities

Associate a probability $p(x)$ with each character x

- $p(x)$ is the probability that a randomly selected character will be x

Probabilities for all characters in alphabet total 1

$$\sum_{x \in \Sigma} p(x) = 1$$

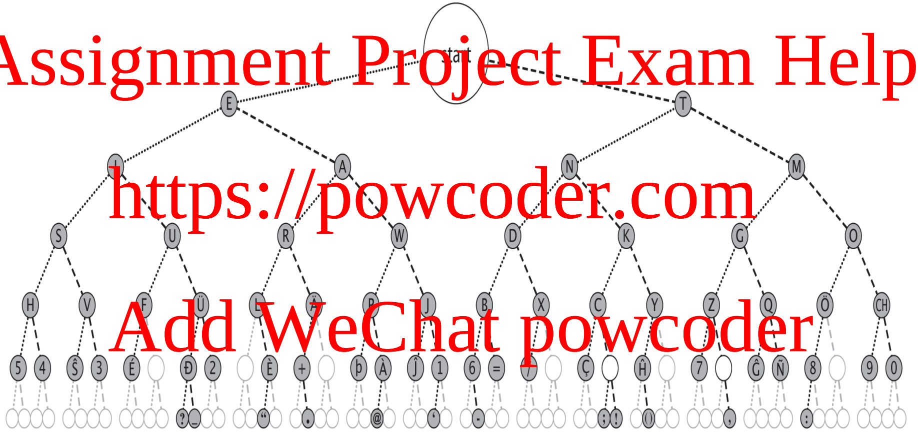
Variable Length Encoding

Underlying principle:

Let more common characters have shorter encodings than less common characters

- This will result in shorter messages on average
- Morse code takes this approach

Morse Code


start
E T
I A N M
S U R W D K G O
H V F Ü L Ä P J B X C Y Z Q Ö CH
5 4 Ŝ 3 É Đ 2 È + Þ À Ĵ 1 6 = / Ç Ĥ 7 Ĝ Ñ 8 9 0
? _ “ . @ ' - ; ! () , :

Average Character Length

Question: When is one encoding preferable to another?

Answer: It depends on the probability distribution of the characters

We want to minimise the **average bits per letter**

$$\mathbf{ABL} = \sum_{x \in \Sigma} p(x) \cdot |\gamma(x)|$$

- The **weighted** average of character encoding length
- Weight is the probability of the character

Common Prefix Problem

Morse Code is potentially ambiguious

- Entire encoding of some characters could be the start of others
- Consider the transmission: - - . .
- This could be: TTEE, or TTI, or TD, etc
- Morse code uses a space (pause) to separate characters

Prefix Codes

Constraint of possible encodings:

- No encoding should be a prefix of any another
- Decoding can then be done without need to mark end of character

'Eager' decoding is safe:

- Consider bits from left to right
- As soon as a sequence of bits $\gamma(x)$ is found for some $x \in \Sigma$, decode that bit string as x
- Consider remaining bit string starting with the next bit

Example

Compare the following two encoding:

Prefix Code

x	$\gamma(x)$
a	00
b	01
c	100
d	101
e	110
f	111

Non-Prefix Code

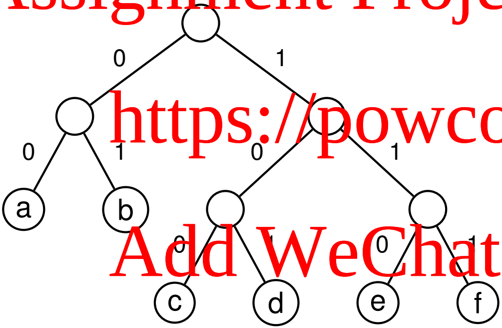
x	$\gamma(x)$
a	0
b	1
c	00
d	01
e	10
f	11

Consider decoding:

0010000110

Prefix Code Trees

Every prefix code can be expressed as a tree



x	$\gamma(x)$
a	00
b	01
c	100
d	101
e	110
f	111

Characters appear **only** at the leaves

Efficient Prefix Code

Note that the depth of x in a prefix code tree equals $|\gamma(x)|$

Property of good prefix code trees:

$$|\gamma(x)| < |\gamma(y)| \quad \text{implies} \quad p(x) \geq p(y)$$

Otherwise, it would be better to swap codes for x and y

Bottom-up Prefix Code Construction

a	b	e	j	s	t	v
0.12	0.10	0.28	0.06	0.23	0.18	0.03

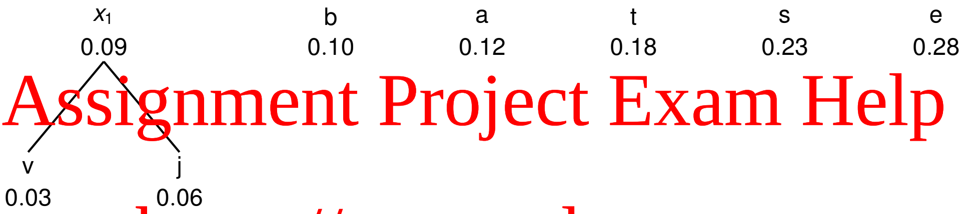
Order alphabet in increasing order of probability

Bottom-up Prefix Code Construction

v	j	b	a	t	s	e
0.03	0.06	0.10	0.12	0.18	0.23	0.28

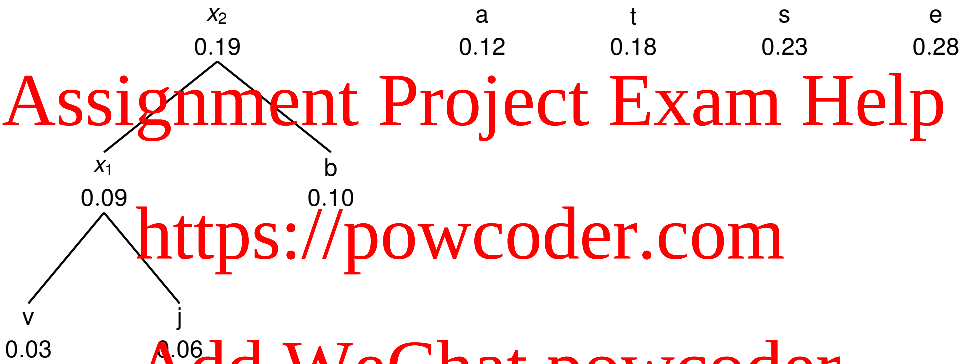
Combine first two characters

Bottom-up Prefix Code Construction



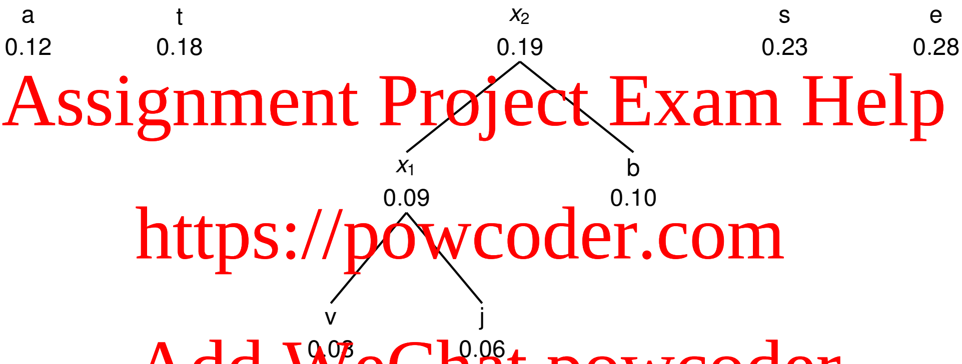
Now left with identical *type* of problem to solve

Bottom-up Prefix Code Construction



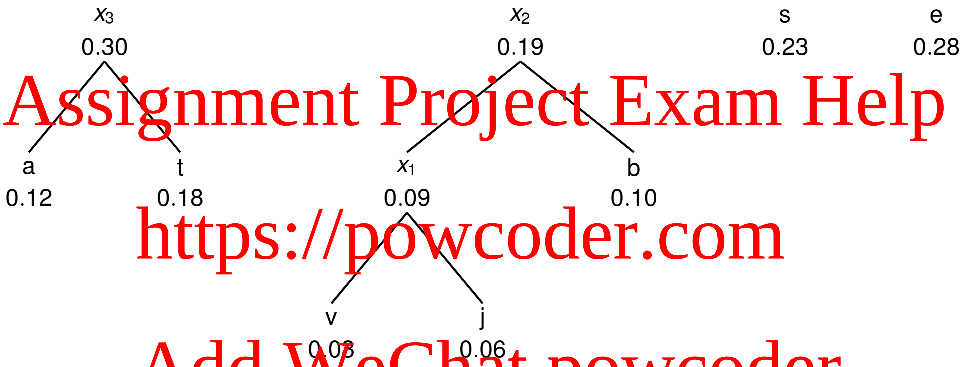
This time we need to re-order the characters

Bottom-up Prefix Code Construction



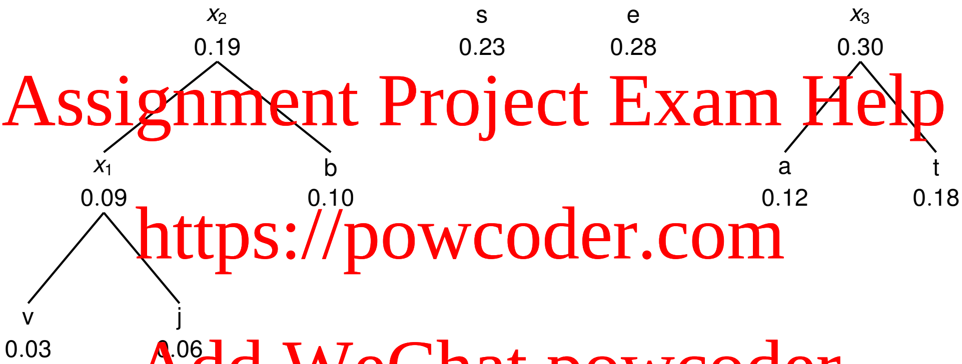
Now we combine the first two

Bottom-up Prefix Code Construction



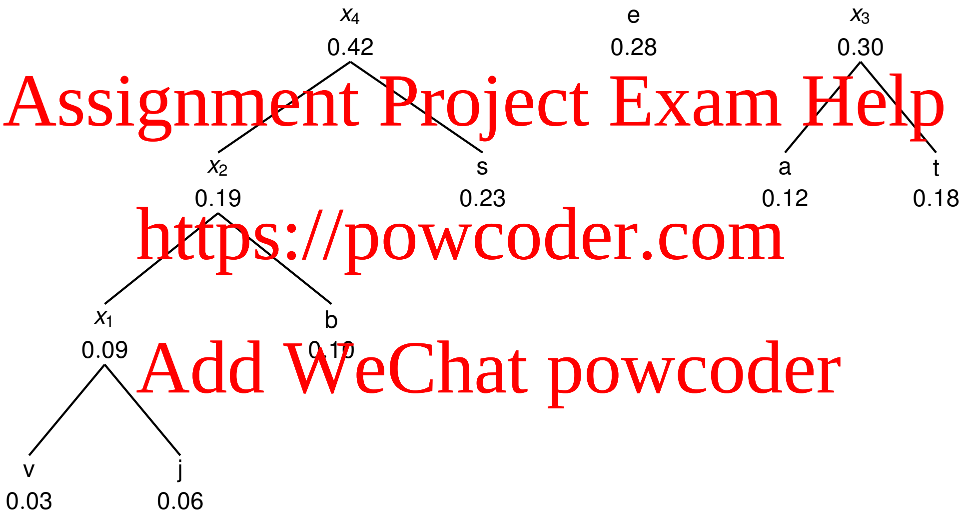
Again we must re-order

Bottom-up Prefix Code Construction

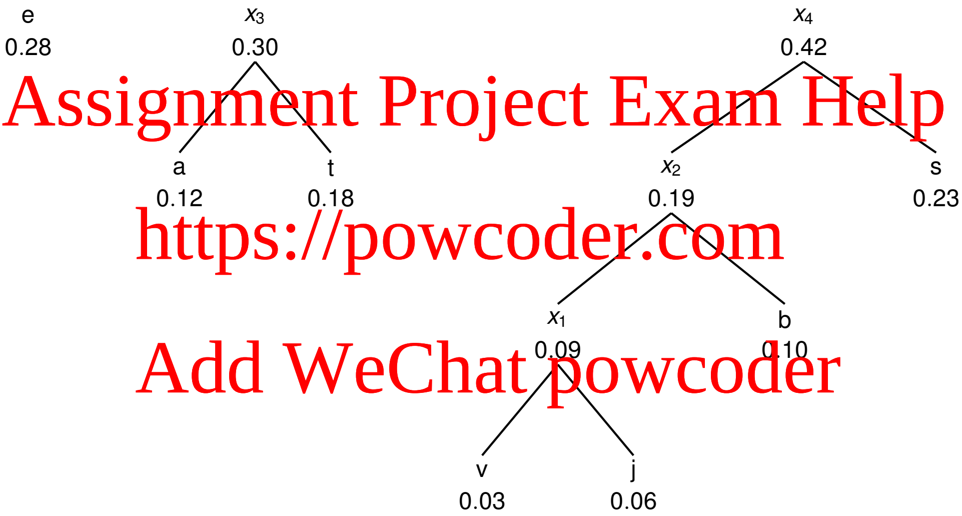


And so on ...

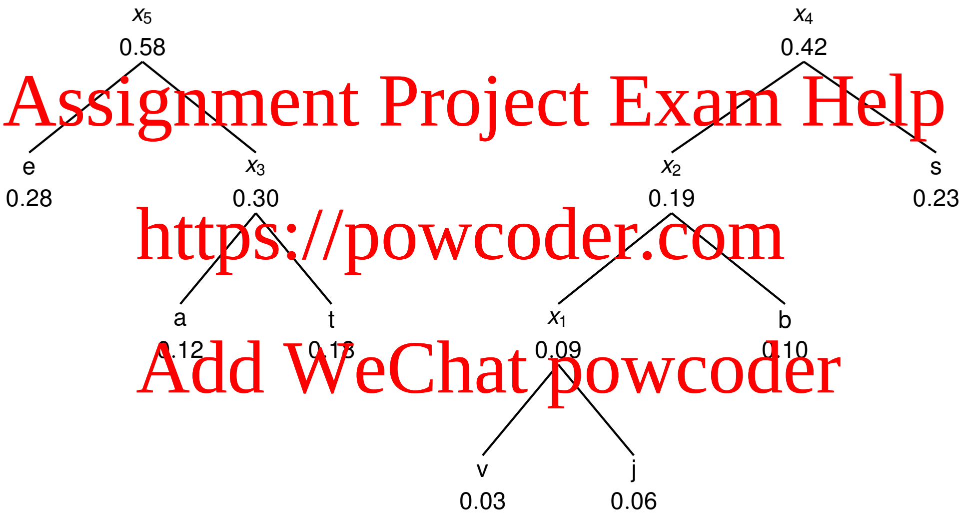
Bottom-up Prefix Code Construction



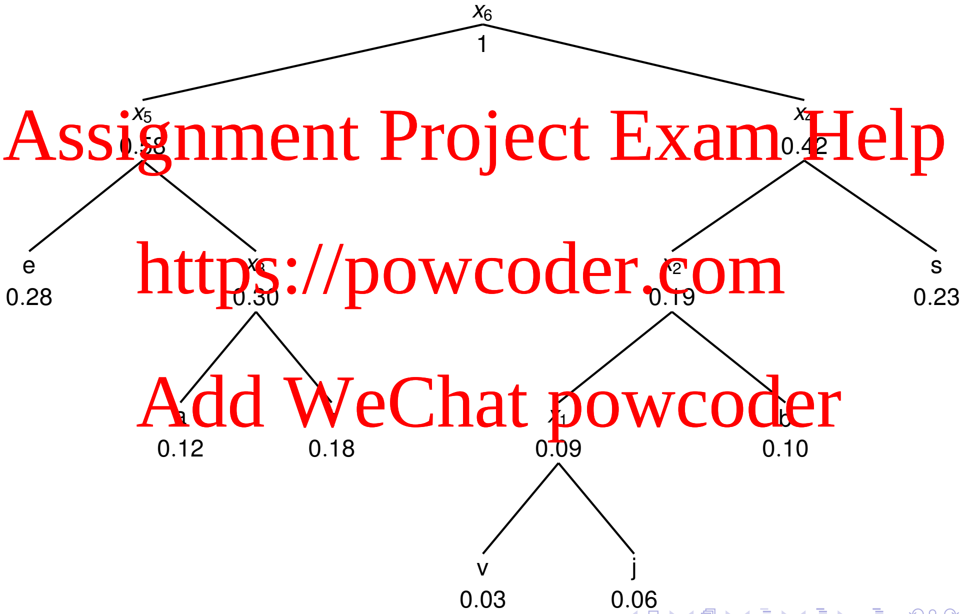
Bottom-up Prefix Code Construction



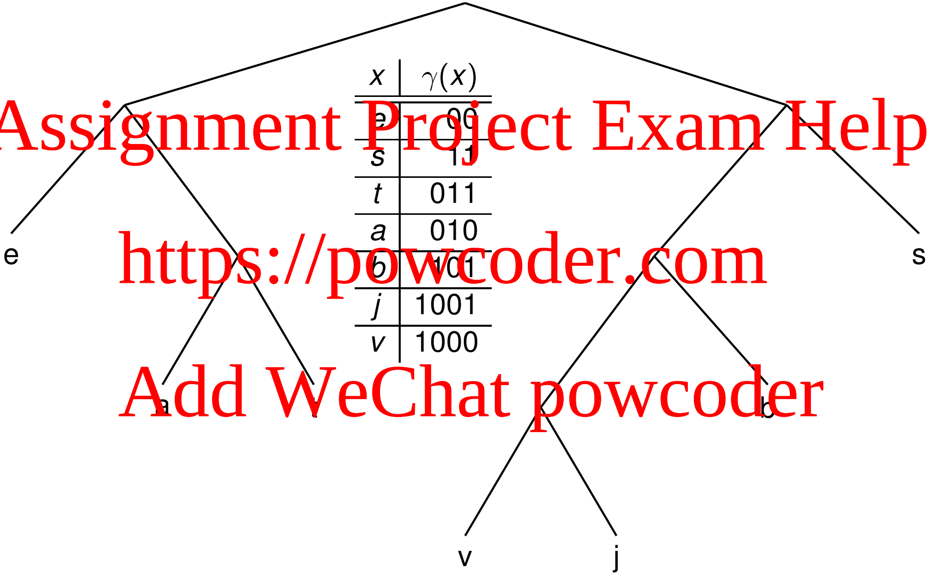
Bottom-up Prefix Code Construction



Bottom-up Prefix Code Construction



Bottom-up Prefix Code Construction



x	$\gamma(x)$
e	00
s	11
t	011
a	010
b	101
j	1001
v	1000

Quality of the Prefix Code

Questions: What is the Average Bits per Letter (ABL) of this code?

Answer:

$$\textbf{ABL} = \sum_{x \in \Sigma} p(x) \cdot |\gamma(x)|$$

$$0.28 \cdot 2 + 0.12 \cdot 3 + 0.18 \cdot 3 + 0.03 \cdot 4 + 0.06 \cdot 4 + 0.10 \cdot 3 + 0.23 \cdot 2$$

$$= 2.58$$

With fixed length encoding 7 characters needs 3 bits per letter

Huffman Code Algorithm

Huffman(Σ, P):

  let Q *be a priority queue of* $x \in \Sigma$ *prioritized by lowest* $P(x)$

  while $|Q| > 1$

    remove the first two items x *and* y *from* Q

    let z *be a new character not in* Σ

    let z *be the parent of* x *and* y

    let $P(z) = P(x) + P(y)$

    add z *to* Q

Correctness of Algorithm

- Algorithm terminates because Q reduces in size by 1 at each iteration
- We need to achieve the following:

$$|\gamma(x)| < |\gamma(y)| \quad \text{implies} \quad p(x) \geq p(y)$$

- Order in which algorithm considers characters guarantees this
- Characters considered later cannot end up deeper in tree

Efficiency of Huffman Code Algorithm

- Let $|\Sigma| = k$
- Creating Q takes $O(k \log k)$ time
- While loop executed $k - 1$ times
- $O(\log k)$ to remove each item from Q
- Assumes Q implemented as a *heap*
- Total running time $O(k \log k)$

Problem for you

Example scenario

x	$p(x)$
e	0.40
i	0.20
o	0.18
f	0.15
k	0.05
z	0.02

- How many bits would be needed without compression?

Problem for you

Example scenario

x	$p(x)$
e	0.40
i	0.20
c	0.18
f	0.15
k	0.05
z	0.02

- How many bits would be needed without compression?

Without compression, 6 characters uses 3 bits since

$$2^2 < 6 \leq 2^3$$

Problem for you

Example scenario

x	$p(x)$
e	0.40
i	0.20
c	0.18
f	0.15
k	0.05
z	0.02

- Apply the algorithm to this case

Problem for you

Example scenario

x	p(x)
e	0.40
i	0.20
c	0.18
f	0.15
k	0.05
z	0.02

- Apply the algorithm to this case

$\gamma(e) = 1$, $\gamma(i) = 001$, $\gamma(c) = 000$, $\gamma(f) = 011$, $\gamma(k) = 0101$, $\gamma(z) = 0100$

Problem for you

Example scenario

x	$p(x)$
e	0.40
i	0.20
c	0.18
f	0.15
k	0.05
z	0.02

- Find the ABL of the resulting code

Problem for you

Example scenario

x	$p(x)$
e	0.40
i	0.20
c	0.18
f	0.15
k	0.05
z	0.02

- Find the ABL of the resulting code

$$0.4 \times 1 + 0.2 \times 3 + 0.18 \times 3 + 0.15 \times 3 + 0.05 \times 4 + 0.02 \times 4 = 2.27$$