JIKSTRA'S ALGORITHM

Correctness proof:

In this algorithm, we maintain two disjoint subsets of V (where V = set of vertices in the graph).
We will call these subsets as S1 and V − S1.

In every vertex \( u \in S \), we maintain a record of the shortest path distance \( d(u) \). Initially \( S = \{s\} \) where \( s = \) source vertex and \( d(s) = 0 \).

For every node \( v \in V - S \), we will calculate the shortest path that can be obtained by travelling through vertices in \( S \) followed by some single edge \( uv \) where \( u \in S \) is the last vertex along that path that lay inside \( S \).

We want to prove: \( \forall u \in S, P_u \) is the shortest path from \( s \) to \( u \).

Proof: We will use mathematical induction.

If \( |S| = 1 \), we have \( d(s) = 0 \) and \( P_s = \{s\} \) with \( S = \{s\} \). This is the base case.

Induction hypothesis: If \( |S| = k \), \( k \geq 1 \), then \( P_u \) is the shortest path from \( s \) to \( u \) for every \( u \in S \).

Induction step: We want to prove this holds...
true when $|S| = k+1$. Set us suppose that in
Dijkstra's algorithm, we added a vertex $v$ to $S$ to increase its
size from $k$ to $k+1$. Then let $(u,v)$ be the
final edge on our $s-v$ path $p_v$.

By induction hypothesis, $P_u$ is the shortest
$s-v$ path which is different from $p_v = [P_u (u,v)]$.

Let's suppose this $s-v$ path leaves $S$ at
vertex $x$. Let $y$ be the last vertex on
this $s-v$ path where $y \notin S$. We want to
show that this $s-v$ path is at least as long as
(or longer than) $p_v = [P_u (u,v)]$.

Now in iteration $k+1$, Dijkstra's algorithm
favored vertex $v$ over vertex $y$.

Thus there is no
because $d(v) \leq d(y)$. Thus there is no
path from $S$ of the form $(s \to x, x \to y, y \to v)$ which is shorter than $p_v$.

This is because $d(v) = |P_v| \leq d(y) = |P_y|$. 
As $d(y) > d(v)$, we must have $d(y) + |y-v| > d(v)$ if $|y-v|$ has a positive weight. If $|y-v|$ has a negative weight, this proof breaks down.

Example to illustrate breakdown of Dijkstra's algorithm if there are negative weights.

![Graph Diagram]

Initially, $S = \{A\}$, $d(A) = 0$, $V-S = \{BCDE\}$.

$\rightarrow$ add in $E$. So $S = \{AE\}$, $V-S = \{BCDF\}$

$\quad d(E) = -5$

$\rightarrow$ add in $F$. So $S = \{AEF\}$, $V-S = \{BCD\}$

$\quad d(F) = -7$

$\rightarrow$ add in $D$. So $S = \{AEFD\}$, $V-S = \{BC\}$

$\quad d(D) = -8$

$\rightarrow$ add in $C$. So $S = \{AEFDC\}$, $V-S = \{B\}$

$\quad d(C) = -18$

$\rightarrow$ add in $B$. So $S = \{AEFDCB\}$, $V-S = \{\}$

$\quad d(B) = -19$
Realize that Dijkstra's algorithm predicts \( d(E) = -5 \) and \( AE \) to be the shortest path, though the true shortest path is \( ABCDFE \) with length \(-13\).

Another example.

\[
\begin{array}{ccc}
A & \xrightarrow{2} & B \\
\downarrow & & \downarrow \\
C & \xrightarrow{-2} & B \\
\end{array}
\]

\[A = \text{source} \]
\[C = \text{dest.} \]
\[d(A) = 0 \quad S = \{A\} \quad V - S = \{B,C\} \]

Choose \( B \) as \( d(B) = 2, \quad d(C) = 3 \).

\[S = \{A,B\} \quad V - S = \{C\}.\]

Now we have to pick \( C \) with \( d(C) = 3 - 2 = 1 \) even though the real shortest path is \( A \rightarrow C \). Next time \( C \) is picked and we terminate. Notice that the real shortest path from \( A \) to \( B \) has length \( 3 + (-2) = 1 \), but Dijkstra's method never chooses it.

Consider a graph with (some or all) negative edge weights. A good question is this: suppose we deduct from all
the edge weights the value of the lowest edge weight (note: this is a -ve number), we now get a graph with all non-negative edge weights. We could now apply Dijkstra's algorithm on this new graph and get a shortest path between any two nodes. However, this shortest path may not correspond to the shortest path in the original graph!

Here is an example

\[ \begin{array}{c}
\text{V1} \quad -5 \\
\text{V3} \quad -3
\end{array} \quad \begin{array}{c}
\text{V2} \\
\end{array} \]

The shortest path from V1 to V2 is V1 - V3 - V2 (length -3)

\[ \begin{array}{c}
\text{V1} \\
2
\end{array} \quad \begin{array}{c}
\text{V3} \\
2
\end{array} \quad \begin{array}{c}
\text{V2} \\
\end{array} \]

(deducting -5 from every edge)

Now, the shortest path is V1 - V2.
Naive implementation: $O(1V^2)$.

Finding vertex with min. value of "dist"—
Updating neighbors of this vertex: $O(\text{degree}_V)$ if "V" was the chosen vertex from $V$.

Total cost = $O(1V^2) + O(1E1)$
= $O(1V^2)$.

If you used a heap, you can find the
least vertex with least value of "dist"
in time $O(\log V1)$ (this includes time taken
readjust the heap). Then we update the
"dist" values of the neighbors of this chosen
vertex which will take $\log V1$ operations
per neighbor, i.e., $O(\log V1 \times \text{degree}(v))$ time.

Hence total time complexity = $O(1E1 \log V1)$. 