Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to $\text{infinity}$
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

**while priority queue is not empty:**
  dequeue a vertex $v$ with the shortest distance
  **for** each vertex $w$ adjacent to $v$:
    **if** $w$’s distance > ($v$’s distance + weight($v$, $w$)):
      set $w$’s parent to $v$
      set $w$’s distance to $v$’s dist. + weight($v$, $w$)

priority queue: S, A, B, C, D

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>infty</td>
<td>infty</td>
<td>infty</td>
<td>infty</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
- Set all vertices to have parent None.
- Set distance for all vertices to infinity.
- Set distance for source vertex to 0.
- Insert all vertices into a priority queue (by distance, smallest first).

While priority queue is not empty:
- Dequeue a vertex \( v \) with the shortest distance.
- For each vertex \( w \) adjacent to \( v \):
  - If \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \)):
    - Set \( w \)'s parent to \( v \).
    - Set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \)).

Dequeued: S
Adjacent to S:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>infy</td>
<td>infy</td>
<td>infy</td>
<td>infy</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \))):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \))

dequeued: S
adjacent to S: A, B, C, D

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
 & S & A & B & C & D \\
\hline
\text{parent} & \text{None} & \text{None} & \text{None} & \text{None} & \text{None} \\
\text{dist.} & 0 & \text{infty} & \text{infty} & \text{infty} & \text{infty} \\
\hline
\end{array}
\]
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \)):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \))

dequeued: S
adjacent to S: A, B, C, D
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue
(by distance, smallest first).
while priority queue is not empty:
  dequeue a vertex \( v \) with the shortest distance
  for each vertex \( w \) adjacent to \( v \):
    if \( w \)'s distance > (\( v \)'s distance + weight(\( v,w \)):
      set \( w \)'s parent to \( v \)
      set \( w \)'s distance to \( v \)'s dist. + weight(\( v,w \))

dequeued: S
adjacent to S: A, B, C, D

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>infty</td>
<td>infty</td>
<td>infty</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
- set all vertices to have parent None.
- set distance for all vertices to infinity
- set distance for source vertex to 0
- insert all vertices into a priority queue (by distance, smallest first).

**while priority queue is not empty:**
- dequeue a vertex \( v \) with the shortest distance
- for each vertex \( w \) adjacent to \( v \):
  - if \( w \)'s distance > (\( v \)'s distance + weight(\( v,w \))):
    - set \( w \)'s parent to \( v \)
    - set \( w \)'s distance to \( v \)'s dist. + weight(\( v,w \))

**dequeued:** S
**adjacent to S:** A,B,C,D

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>infty</td>
<td>infty</td>
<td>infty</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).
while priority queue is not empty:
  dequeue a vertex \( v \) with the shortest distance
  for each vertex \( w \) adjacent to \( v \):
    if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
      set \( w \)'s parent to \( v \)
      set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))

dequeued: \( S \)
adjacent to \( S \): \( A,B,C,D \)

<table>
<thead>
<tr>
<th></th>
<th>( S )</th>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( D )</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>( S )</td>
<td>( S )</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>infty</td>
<td>infty</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))

dequeued: S
adjacent to S: A, B, C, D

<table>
<thead>
<tr>
<th>parent</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>S</td>
<td>S</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>infty</td>
<td>infty</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \))):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \))

dequeued: S
adjacent to S: A, B, C, D
**Edgar Dijkstra's algorithm:**
- set all vertices to have parent `None`.
- set distance for all vertices to `infinity`.
- set distance for source vertex to `0`.
- insert all vertices into a priority queue (by distance, smallest first).
- while priority queue is not empty:
  - dequeue a vertex `v` with the shortest distance.
  - for each vertex `w` adjacent to `v`:
    - if `w`'s distance > (v's distance + weight(v,w)):
      - set `w`'s parent to `v`.
      - set `w`'s distance to `v`'s dist. + weight(v,w).

**Priority Queue:** A, B, C, D

**Dequeued:** S

**Adjacent to S:** A, B, C, D

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>None</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>infty</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))

dequeued: \( S \)
adjacent to \( S \): A, B, C, D

<table>
<thead>
<tr>
<th>parent</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

priority queue: A, B, C, D
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
  dequeue a vertex \( v \) with the shortest distance
for each vertex \( w \) adjacent to \( v \):
  if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
    set \( w \)'s parent to \( v \)
    set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))

dequeued: S
adjacent to S: A,B,C,D

<table>
<thead>
<tr>
<th>parent</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
    dequeue a vertex $v$ with the shortest distance
    for each vertex $w$ adjacent to $v$:
        if $w$'s distance > ($v$'s distance + weight($v$, $w$)):
            set $w$'s parent to $v$
            set $w$'s distance to $v$'s dist. + weight($v$, $w$)

dequeued:
adjacent to:
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
dequeue a vertex \( v \) with the shortest distance
for each vertex \( w \) adjacent to \( v \):
if \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \))):
set \( w \)'s parent to \( v \)
set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \))

dequeued: A
adjacent to A:
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).
while priority queue is not empty:
  dequeue a vertex \( v \) with the shortest distance
  for each vertex \( w \) adjacent to \( v \):
    if \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \)):
      set \( w \)'s parent to \( v \)
      set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \))

dequeued: A
adjacent to A:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue
(by distance, smallest first).

while priority queue is not empty:
  dequeue a vertex $v$ with the shortest distance
  for each vertex $w$ adjacent to $v$:
    if $w$'s distance > ($v$'s distance + weight($v$, $w$)):
      set $w$'s parent to $v$
      set $w$'s distance to $v$'s dist. + weight($v$, $w$)

dequeued:
adjacent to:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
dequeue a vertex \( v \) with the shortest distance
for each vertex \( w \) adjacent to \( v \):
if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
set \( w \)'s parent to \( v \)
set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))

Dequeued: B
adjacent to B: C, D

<table>
<thead>
<tr>
<th>parent</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

priority queue: B, C, D
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
dequeue a vertex \( v \) with the shortest distance

for each vertex \( w \) adjacent to \( v \):
if \( w \)'s distance > (\( v \)'s distance + weight(\( v,w \))):
set \( w \)'s parent to \( v \)
set \( w \)'s distance to \( v \)'s dist. + weight(\( v,w \))

Dequeued: B
adjacent to B: C,D

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).
while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))

Dequeued: B
adjacent to B: C, D

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue
(by distance, smallest first).
while priority queue is not empty:
   dequeue a vertex v with the shortest distance
   for each vertex w adjacent to v:
      if w’s distance > (v’s distance + weight(v,w)):
         set w’s parent to v
         set w’s distance to v’s dist. + weight(v,w)
Dequeued: B
adjacent to B: C, D

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to \textit{infinity}
set distance for source vertex to 0
insert all vertices into a \textit{priority queue} (by distance, smallest first).

\textbf{while priority queue is not empty:}
  dequeue a vertex \(v\) with the shortest distance
  for each vertex \(w\) adjacent to \(v\):
    if \(w\)'s distance > (\(v\)'s distance + weight(\(v,w\)):
      set \(w\)'s parent to \(v\)
      set \(w\)'s distance to \(v\)'s dist. + weight(\(v,w\))

\begin{itemize}
  \item Dequeued: B
  \item adjacent to B: C,D
\end{itemize}

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>B</td>
<td>S</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue
(by distance, smallest first).
while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \))):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \))

Dequeued: B
adjacent to B: C, D
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)’s distance + weight(\( v, w \)):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)’s dist. + weight(\( v, w \))

Dequeued: B
adjacent to B: C, D

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
    dequeue a vertex \( v \) with the shortest distance
    for each vertex \( w \) adjacent to \( v \):
        if \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \)):
            set \( w \)'s parent to \( v \)
            set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \))

Dequeued: B
adjacent to B: C, D
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
  dequeue a vertex \( v \) with the shortest distance
  for each vertex \( w \) adjacent to \( v \):
    if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
      set \( w \)'s parent to \( v \)
      set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))

Dequeued:
adjacent to:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
dequeue a vertex $v$ with the shortest distance
for each vertex $w$ adjacent to $v$:
  if $w$'s distance > ($v$'s distance + weight($v$, $w$)):
    set $w$'s parent to $v$
    set $w$'s distance to $v$'s dist. + weight($v$, $w$)

Dequeued: C
adjacent to C:
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue
(by distance, smallest first).

while priority queue is not empty:
  dequeue a vertex \( v \) with the shortest
distance

for each vertex \( w \) adjacent to \( v \):
  if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
    set \( w \)'s parent to \( v \)
    set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
  dequeue a vertex v with the shortest distance
  for each vertex w adjacent to v:
    if w's distance > (v's distance + weight(v,w)):
      set w's parent to v
      set w's distance to v's dist. + weight(v,w)

Dequeued:
adjacent to:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
dequeue a vertex \( v \) with the shortest distance
for each vertex \( w \) adjacent to \( v \):
if \( w \)'s distance > (\( v \)'s distance + weight(\( v, w \)):
    set \( w \)'s parent to \( v \)
    set \( w \)'s distance to \( v \)'s dist. + weight(\( v, w \))

Dequeued: D
adjacent to D: A

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).
while priority queue is not empty:
  dequeue a vertex $v$ with the shortest distance
  for each vertex $w$ adjacent to $v$:
    if $w$'s distance > ($v$'s distance + weight($v$, $w$):
      set $w$'s parent to $v$
      set $w$'s distance to $v$'s dist. + weight($v$, $w$)

Dequeued: D
adjacent to D: A
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

**while priority queue is not empty:**

- dequeue a vertex \( v \) with the shortest distance
- for each vertex \( w \) adjacent to \( v \):
  - if \( w \)'s distance > (\( v \)'s distance + weight(\( v,w \))):
    - set \( w \)'s parent to \( v \)
    - set \( w \)'s distance to \( v \)'s dist. + weight(\( v,w \))

### Dequeued: D
adjacent to D: A

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
  dequeue a vertex \( v \) with the shortest distance
  for each vertex \( w \) adjacent to \( v \):
    if \( w \)'s distance > (\( v \)'s distance + weight(\( v \),\( w \))):
      set \( w \)'s parent to \( v \)
      set \( w \)'s distance to \( v \)'s dist. + weight(\( v \),\( w \))

Dequeued: D
adjacent to D: A
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
- set all vertices to have parent None.
- set distance for all vertices to infinity
- set distance for source vertex to 0
- insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
  dequeue a vertex $v$ with the shortest distance
  for each vertex $w$ adjacent to $v$:
    if $w$'s distance > ($v$'s distance + weight($v$, $w$)):
      set $w$'s parent to $v$
      set $w$'s distance to $v$'s dist. + weight($v$, $w$)

Dequeued:
adjacent to:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm for weighted graphs

Edgar Dijkstra's algorithm:
set all vertices to have parent None.
set distance for all vertices to infinity
set distance for source vertex to 0
insert all vertices into a priority queue (by distance, smallest first).

while priority queue is not empty:
  dequeue a vertex v with the shortest distance
  for each vertex w adjacent to v:
    if w's distance > (v's distance + weight(v,w)):
      set w's parent to v
      set w's distance to v's dist. + weight(v,w)

STOP
Dijkstra's algorithm for weighted graphs

The table is ready to be used.

For example, the shortest path from S to D is S → B → D

the shortest path from S to C is S → B → C

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>None</td>
<td>S</td>
<td>S</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>dist.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>