- A greedy technique that actually works well
- Starting from a certain vertex \( v_0 \), we set up a Map to hold all the running min costs \( D_i \) from \( v_0 \) to various destinations \( v \).
- Like the unweighted case, we have an “eyeball” that indicates what node we are visiting.
- We move the eyeball to a new node \( v \), the one with the least cost from \( v_0 \) that has not been visited by the eyeball.
We compute the min cost $D_w$ from $v_0$ to various destinations $w$ following any path through the visited node $v$

- $D_w = \min\{D_w, D_v + c_{vw}\}$
- $c_{vw}$ = cost for edge $v \rightarrow w$ (these are all constant positive numbers)

Repeat by moving the eyeball to another node which has the lowest cost and has not been visited.
Adjust $D_w$

The eyeball is at $v$ and $w$ is adjacent to $v$, so $D_w$ should be lowered from 8 to 6.

**Figure 14.23**
The eyeball is at $v$ and $w$ is adjacent, so $D_w$ should be lowered to 6.
Running Example, Starting From $V_0$
Running Example, Starting From $V_0$
Path with Minimum Weights from \( V_0 \)

![Graph Image]

<table>
<thead>
<tr>
<th>eyeball</th>
<th>( D_0 )</th>
<th>( D_1 )</th>
<th>( D_2 )</th>
<th>( D_3 )</th>
<th>( D_4 )</th>
<th>( D_5 )</th>
<th>( D_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-)</td>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>( V_0 )</td>
<td>—</td>
<td>2</td>
<td>( \infty )</td>
<td>1</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>( V_3 )</td>
<td>—</td>
<td>2</td>
<td>3</td>
<td>—</td>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>( V_1 )</td>
<td>—</td>
<td>—</td>
<td>3</td>
<td>—</td>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>( V_4 )</td>
<td>—</td>
<td>—</td>
<td>3</td>
<td>—</td>
<td>—</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>( V_2 )</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>( V_6 )</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>( V_5 )</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>final:</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>
Pseudocode for Dijkstra’s Algorithm

Given a directed graph G and source node S.
Set up storage for a distance for each node, \( D_i \):
   For each node i, set \( D_i = c_{S,i} \)
   (cost of the edge from S to i)
Set up a data structure P for unvisited nodes and put all nodes except S in it.
Loop while P is not empty
   Choose a node v in P with minimum \( D_i \)
   Delete v from P
   (it’s now visited, and now the ‘‘eyeball’’ node)
Loop though nodes w adjacent to v and in P
   \( D_w = min(D_w, D_v + c_{vw}) \)
A simple implementation would mimic our manual processing of the D’s earlier:

- scan the D’s to choose a new eyeball, $O(|V|)$ for each node, or $O(|V|^2)$ in all, and then scan the edges from the eyeball node for D-updates, for each eyeball node, $|E|$ in all, so total $O(|V|^2)$, since $|E| < |V|^2$.

- Not too bad, but not great.

- But we need the minimal D each time. How can we do better?

**Using Priority Queue to find min: Change from $|E|$ to Log$|E|$.**

The are $|E|$ insertions to the PQ, thus $O(|E| \cdot \text{Log}|E|)$. 